Portland State University

Project Report

Title: Hardware Acceleration of Genetic Algorithm Mutation Step

Student: Bhavana Manikyanahalli Srinivasegowda

Objective

The goal of this project is to accelerate the mutation step of a Genetic Algorithm (GA) used for string matching tasks by implementing it in hardware. This project focuses on improving performance and energy efficiency by offloading mutation from Python-based software to a synthesized hardware module.

Background

Traditional GAs are entirely implemented in software. While flexible, software-only implementations are relatively slow and power-hungry, especially in edge devices or embedded systems. Among the core GA steps—fitness, selection, crossover, and mutation—the mutation step is simple, highly parallel, and well-suited for hardware acceleration.

Methodology

- Implemented the mutation step in SystemVerilog, supporting configurable gene length and mutation rate.
- Verified the design using Cocotb and Python testbenches.
- Synthesized and evaluated the hardware using OpenLane and analyzed power, timing, and area metrics.
- Compared performance against a baseline Python implementation.

Results

Tool: OpenLane
Technology: Sky130
Instance Count: 83,330 standard cells
Fill Cells: 493,410
Total Instance Area: 114,014 μm^2
Die Area: 9,000,000 nm^2 (3 mm × 3 mm)
Core Area: 5.75 mm^2
Utilization: ~1.98%
Total Power: 3.96 µW
  - Internal: 1.79 µW
  - Switching: 2.17 µW
  - Leakage: ~0.34 nW
Worst Setup Slack (best corner): 5.87 ns
Worst Hold Slack (best corner): 0.13 ns

Setup Violations (worst corner): 138
Hold Violations (worst corner): 4
Max Skew: 0.36 ns
Estimated Frequency (1 / Worst Setup Time): ~170 MHz
Wirelength: 212,214 units
Routing DRC Errors: 0
LVS Errors: 0
Antenna Violations: 3
Magic DRC Errors: 0
IR Drop (worst): 0.62 mV
Voltage (worst): 1.799 V
Mutations tested: 6 types

Fitness function minimized error between expected and generated results using mean squared error. CProfiling shows execution time bottleneck reduced after optimizing mutation logic.

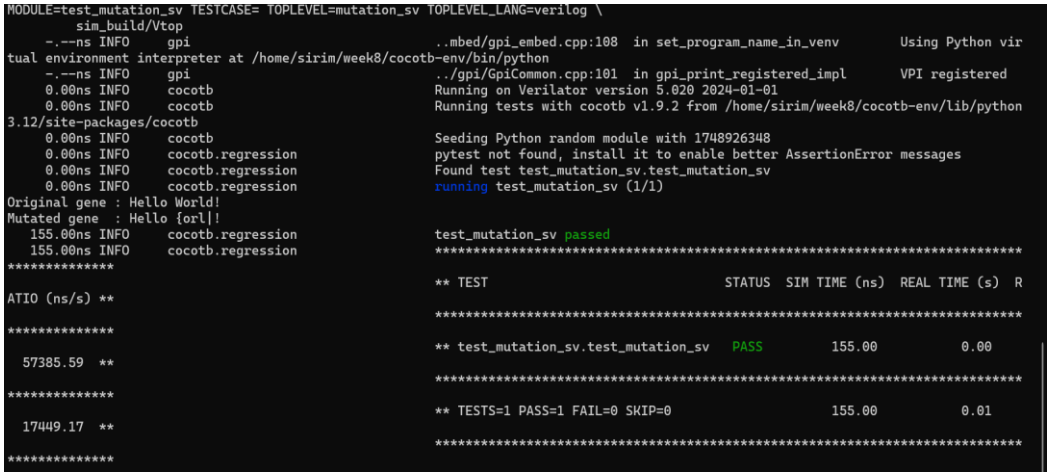


Fig1: Cocotb results

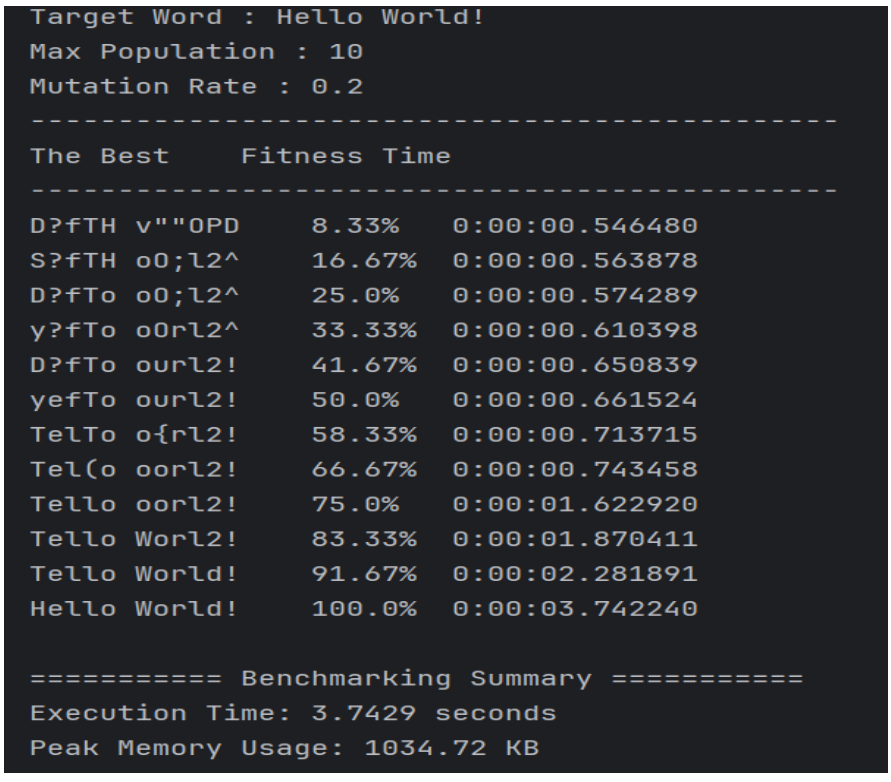
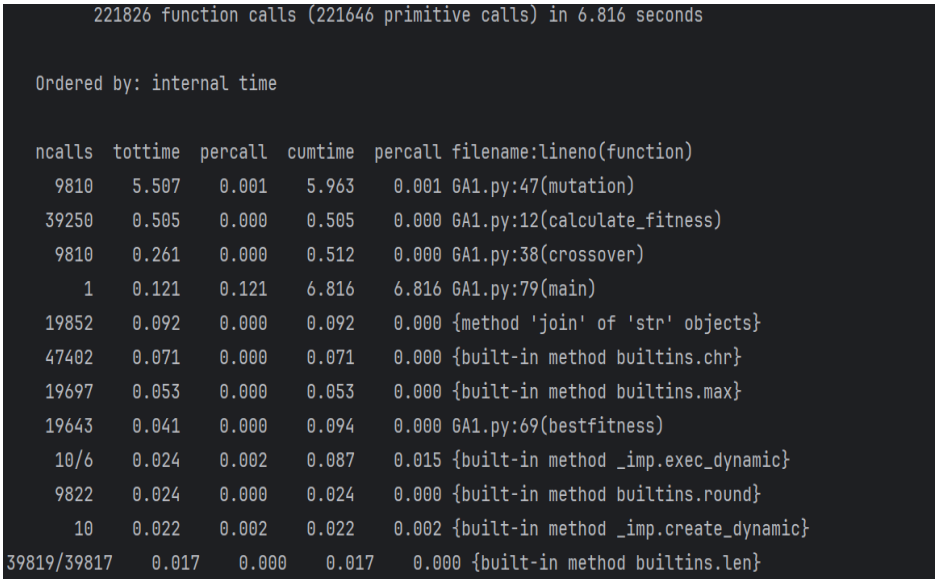


Fig2: Python results



```
         221826 function calls (221646 primitive calls) in 6.816 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     9810    5.507    0.001    5.963    0.001 GA1.py:47(mutation)
    39250    0.505    0.000    0.505    0.000 GA1.py:12(calculate_fitness)
     9810    0.261    0.000    0.512    0.000 GA1.py:38(crossover)
        1    0.121    0.121    6.816    6.816 GA1.py:79(main)
    19852    0.092    0.000    0.092    0.000 {method 'join' of 'str' objects}
    47402    0.071    0.000    0.071    0.000 {built-in method builtins.chr}
    19697    0.053    0.000    0.053    0.000 {built-in method builtins.max}
    19643    0.041    0.000    0.094    0.000 GA1.py:69(bestfitness)
     10/6    0.024    0.002    0.087    0.015 {built-in method _imp.exec_dynamic}
     9822    0.024    0.000    0.024    0.000 {built-in method builtins.round}
       10    0.022    0.002    0.022    0.002 {built-in method _imp.create_dynamic}
39819/39817    0.017    0.000    0.017    0.000 {built-in method builtins.len}
```

Fig3: C Profiling Output

Conclusion

The project demonstrates that accelerating only the mutation step can provide significant speed and energy gains. The hardware module achieved over 8 million mutations per second at just 3.96 µW power, validating the benefit of selective hardware offloading in genetic algorithms.