

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 \times 3 mm)

Core Area: 5.75 mm^2

Utilization: $\sim 1.98\%$

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.

```

MODULE=test_mutation_sv TESTCASE= TOPLEVEL=mutation_sv TOPLEVEL_LANG=verilog \
sim_build/Vtop
--ns INFO gpi .mbed/gpi_embed.cpp:108 in set_program_name_in_venv Using Python vir
tual environment interpreter at /home/sirim/week8/cocotb-env/bin/python
--ns INFO gpi .mbed/gpi_embed.cpp:101 in gpi_print_registered_impl VPI registered
0.00ns INFO cocotb Running on Verilator version 5.020 2024-01-01
0.00ns INFO cocotb Running tests with cocotb v1.9.2 from /home/sirim/week8/cocotb-env/lib/python
3.12/site-packages/cocotb
0.00ns INFO cocotb Seeding Python random module with 1748926348
0.00ns INFO cocotb.regression pytest not found, install it to enable better AssertionError messages
0.00ns INFO cocotb.regression Found test test_mutation_sv.test_mutation_sv
0.00ns INFO cocotb.regression running test_mutation_sv (1/1)
Original gene : Hello World!
Mutated gene : Hello {orl|}
155.00ns INFO cocotb.regression test_mutation_sv passed
155.00ns INFO cocotb.regression
*****
** TEST STATUS SIM TIME (ns) REAL TIME (s) R
*****
** test_mutation_sv.test_mutation_sv PASS 155.00 0.00
*****
** TESTS=1 PASS=1 FAIL=0 SKIP=0 155.00 0.01
*****
*****
ATIO (ns/s) **
*****
57385.59 **
*****
17449.17 **
*****

```

Fig1: Cocotb results

```

Target Word : Hello World!
Max Population : 10
Mutation Rate : 0.2
-----
The Best      Fitness Time
-----
D?fTH v""OPD      8.33%    0:00:00.546480
S?fTH o0;l2^      16.67%    0:00:00.563878
D?fTo o0;l2^      25.0%     0:00:00.574289
y?fTo o0rl2^      33.33%    0:00:00.610398
D?fTo ourl2!      41.67%    0:00:00.650839
yefTo ourl2!      50.0%     0:00:00.661524
TelTo o{rl2!      58.33%    0:00:00.713715
Tel(o oorl2!      66.67%    0:00:00.743458
Tello oorl2!      75.0%     0:00:01.622920
Tello Worl2!      83.33%    0:00:01.870411
Tello World!      91.67%    0:00:02.281891
Hello World!     100.0%    0:00:03.742240

===== Benchmarking Summary =====
Execution Time: 3.7429 seconds
Peak Memory Usage: 1034.72 KB

```

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 μ W power, validating the benefit of selective hardware offloading in genetic algorithms.