Evolutionary Hardware with Dynamic Problems

Alexander Dalton, supervised by Dr. Seth Bullock

Introduction

Despite constant advances in machine learning and computer architecture, the intersection of the two remains a relatively unexplored field. One such area, evolutionary hardware, splices FPGA (Field-Programmable Gate Array) architectures and genetic algorithms. Evolutionary pressure is applied to a population of FPGA configurations to find a solution to a given problem. Since early attempts at evolutionary hardware, FPGA technology has improved dramatically, and evolutionary algorithms have seen unprecedented successes in a variety of domains.

Simulated FPGA

An FPGA consists of a uniform mesh of CLBs (Configurable Logic Blocks). These logic blocks take input from their neighboring cells, and send distinct outputs to their neighboring cells. They also have a configurable binary function F. Each output can be independently assigned to send any of the cell's inputs or the output of the function F.

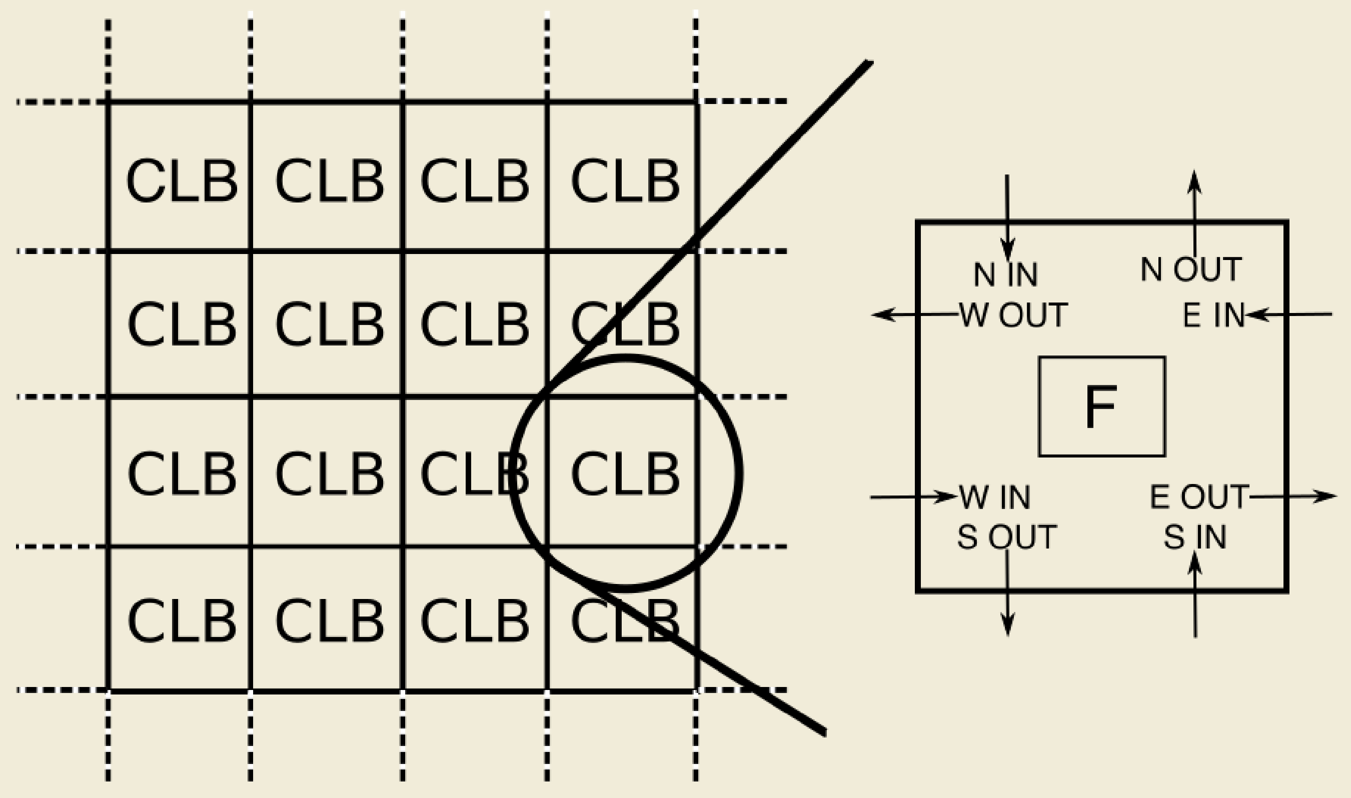

Figure: The structure of an FPGA

To improve development time a bare-bones simulated FPGA was constructed mirroring the description above.

Genetic Algorithm

A genetic algorithm takes a population of binary strings, evaluates them, and generates a new population based on the fitness of the individuals in the previous generation, then with a small probability flips random bits in the new population. An FPGA configuration is encoded as a bitstring where each cell is defined by 16 bits. The fitness of a configuration is the total number of correct output bits.

Multi-objective genetic algorithms were used to improve population diversity and mitigate against evolutionary dead ends. To this end, the distance between an individual and the rest of the population is taken into account when generating a new population.

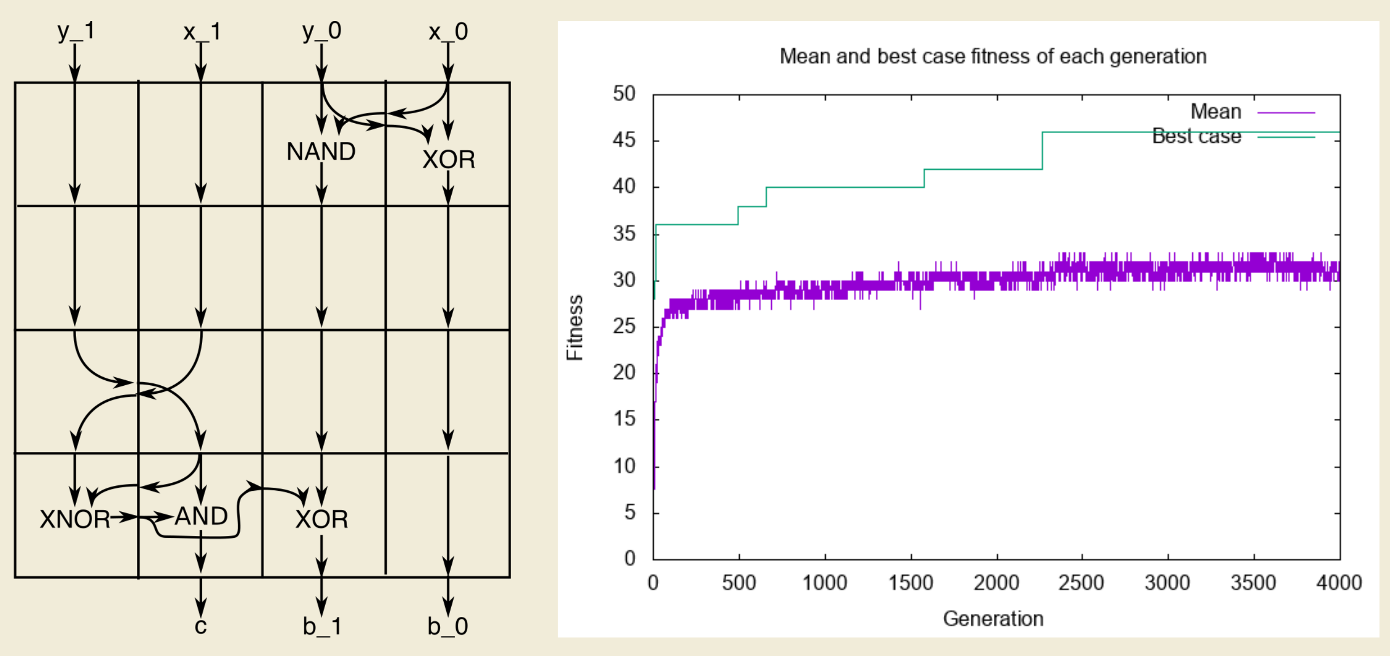

Figure: Left: A diagram of an evolved 2-bit ADDer
Right: The fitness over time of the process which generated the ADDer

Further Work

- Physical implementation
- Scaling experiments
- Optimising a dynamic problem for execution time
- Using coevolution to reduce population disengagement

Fault Tolerance

Evolutionary hardware can be used to recover from otherwise devastating faults. In the example below the outputs of the operation F within the CLB marked in red were clamped to a value representing an "undefined output". Starting with the ideal configuration on the left and activating the fault, the genetic algorithm recognised the current solution as suboptimal and generated the configuration on the right.

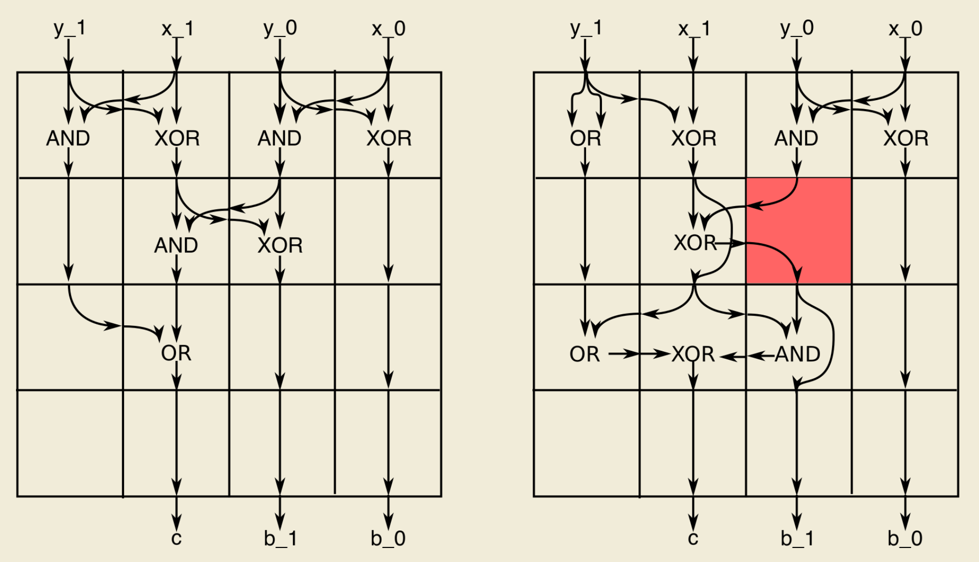

Figure: Left: A perfect ADDer
Right: An evolved solution to a logic fault in a CLB (marked in red)

The graph below on the left shows the recovery time of the perfect ADDer example above.

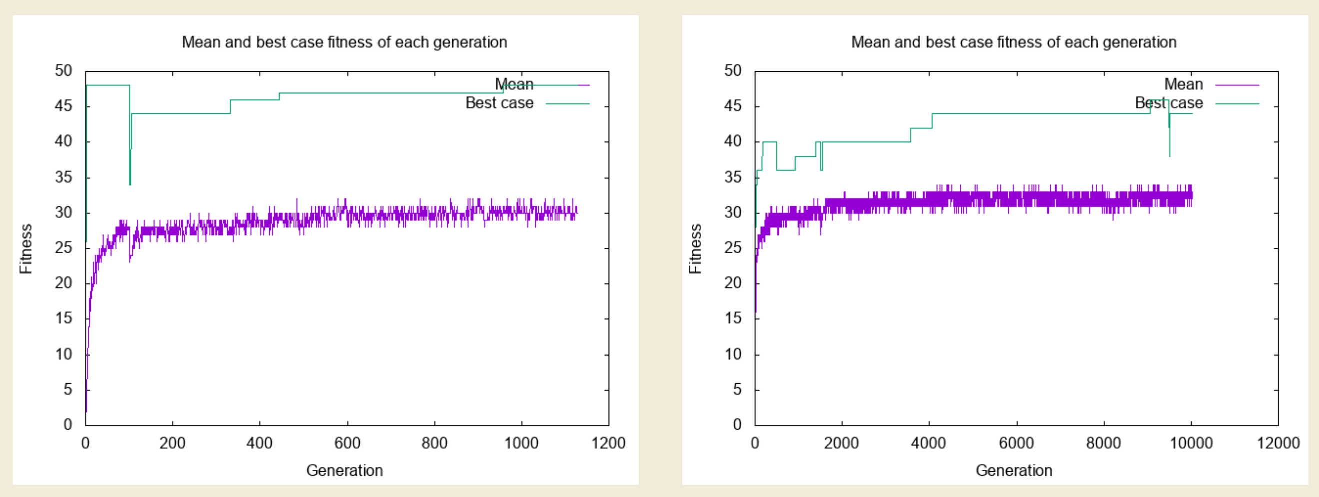

Figure: Left: The fitness over time of a perfect ADDer recovering from a fault
Right: The fitness over time evolving an ADDer with a sticky fault

Simulating a sticky fault by activating and deactivating a fault every 500 generations results in the fitness graph above on the right. When the fault is activated it has an adverse affect on fitness, but a solution is found which works when the fault is both active and inactive.

Dynamic Problem Optimisation

By introducing subtraction as a problem alongside addition, the relative weightings of a correct answer for each problem can be varied to explore dynamically changing problems. In the graph below the genetic algorithm starts with a perfect ADDer and 100% of the fitness weighting assigned to correct ADDs, every 200 generations this shifts by 10% towards SUBs.

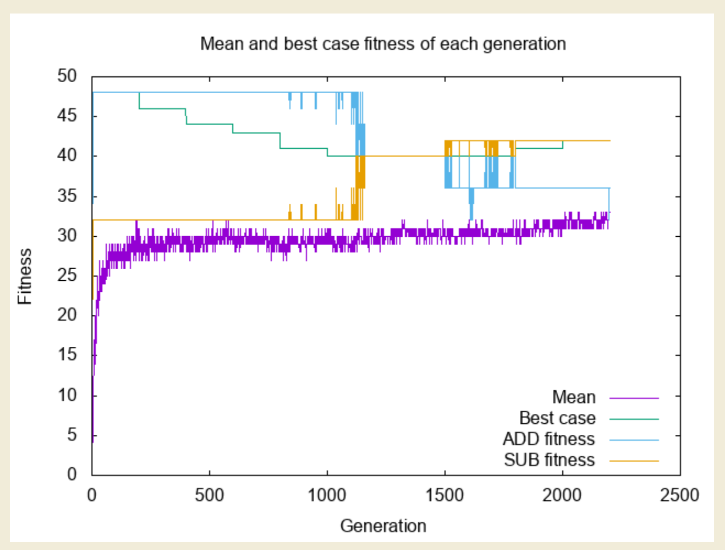

Figure: The fitness over time while the fitness weighting shifts from ADDs to SUBs

At around 1000 generations, when the weightings are equal, the genetic algorithm selects a configuration with equal performance on ADDs and SUBs, a dramatic improvement in SUB performance.

github.com/AlexDalt/MalleableHardwareAdder
ad14810@bristol.ac.uk
University of Bristol, Department of Computer Science