Automatic Creation of Digital Fast Adder Circuits by Means of Genetic Programming

Karim Nassar
Lockheed Martin Missiles and Space
1111 Lockheed Martin Way
Sunnyvale, CA 94089
Karim.Nassar@lmco.com
408-742-9915

Abstract: *This paper explores the use of genetic programming applied to the task of evolving 2-bit and 4-bit fast adder structures using NOT gates and two input AND, OR, & XOR gates. Two genetic programming strategies are applied and their performance is examined.*

Introduction

The successful application of genetic programming to circuit design has been demonstrated on a variety of analog circuits (Koza et al, 1997). This paper explores the ability of genetic programming to evolve basic digital circuits.

Adder circuits are basic building blocks of digital circuits. There are a wide variety of commonly used adder circuits, each possessing unique design tradeoffs in the balance of speed versus gate count. Fast adder circuits trade silicon area (number of gates) for speed (determined by the longest path through the gates) and are essential building blocks in ALUs (Arithmetic Logic Units). Fast adder circuits described in this paper are designed solely for speed with no regard to the number of gates used. Such fast adders are usually only practical for low order adders of five input bits and less. These low-order fast adders are essential building blocks of high order fast and practical adders such as the popular Carry-Save Adder.

Methods

This paper examines two different approaches to the evolution of fast adder circuits. The first approach evolves a two-bit fast adder, otherwise known as a half-adder. The second approach evolves a 4-bit adder.

In this research the following simplifications are made. Logic gates available are limited to the NOT, AND, OR, & XOR. There are several motivations behind this. Although all logic can be expressed in terms of the NOT gate and the AND gate, the OR gate and XOR gate are included because they provide functionality frequently needed in adder circuits. Additionally AND, OR, & XOR form the complete set of non-trivial, non-inverting, Boolean functions of two inputs. Inverting logic is excluded, e.g. NAND, NOR, XNOR, to keep the function set smaller and hence reduce the computational complexity and computer run-time (Koza 1992). Logic gates are limited to two inputs for the same reason.

All gate delays are assumed uniform, as gate delay is a technology specific parameter that cannot be generalized over ASICs, FPGAs, CPLDs, & PLAs.

Table 1: GP Parameters for the First Approach

Objective:	Find a structure whose input is two 2-bit numbers and whose output is the sum expressed as a 3-bit number.
Terminal Set:	A0, B0 for RPB0 A1, A0, B1, B0 for RPB1 and RPB2
Function Set:	NOT, AND, OR, XOR
Fitness Cases:	Sixteen fitness cases covering all possible Boolean input cases
Raw Fitness:	The number of output bits correctly computed over all the fitness cases. (max 48)
Standardized Fitness:	(48 – raw fitness) + max allowed depth (17) if hits < 48 largest RPB depth if hits = 48
Adjusted Fitness:	0.5 * (hits/48) if hits < 48 0.5 + 0.4 * (17 – largest RPB depth)/(17) + 0.1 * (3*17 - sum of tree depths of the RPBs)/(3*17) if hits = 48
Hits:	Same as Raw Fitness
Parameters:	M = 2000 (with overselection) G = 201
Success Predicate:	None. Hits = 48 indicates a perfect adder, but does not terminate run.
Rules of Construction:	There are three RPBs, one for each resultant bit

The first approach uses the parameters in Table 1 for the evolution of a 2-bit adder. The number of terminals and fitness cases are scaled up for a 4-bit adder. A graphical depiction of the multiple RPBs, Result Producing Branches, structure for a 2-bit adder is shown in Figure 1.

In the first approach the adjusted fitness measure (ranging from 0 to 1, 1 being perfect) is carved into three unequal parts. One-half of the adjusted fitness measures the correctness of the adder over the 16 fitness cases. If the adder performs flawlessly over the fitness cases then the next two parts of the fitness measure are evaluated, otherwise they contribute zero to the fitness measure. Forty percent of the fitness measure is to reward perfect adders with the least delay. The remaining 10% of the adjusted fitness is to reward solutions that have generally short delays, e.g. favor a solution with depths of {3, 3, 1} over a solution with depths of {3, 3, 2}.

Crossover only occurs with like RPBs.

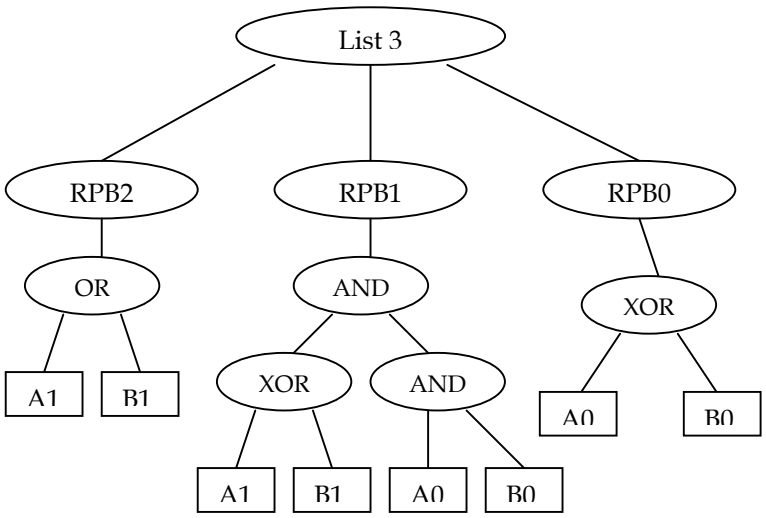

Figure 1: Illustration of First Approach on a 2-bit Adder

Table 2: GP Parameters for the Second Approach

Objective:	Find a structure whose input is two 4-bit numbers and whose output is the highest bit of the sum.
Terminal Set:	A3, A2, A1, A0, B3, B2, B1, B0
Function Set:	NOT, AND, OR, XOR
Fitness Cases:	256 fitness cases covering all possible Boolean input cases
Raw Fitness:	The number of output bits correctly computed over all the fitness cases. (max 256)
Standardized Fitness:	(256 – raw fitness) + max allowed depth (17) if hits < 256 largest RPB depth if hits = 256
Adjusted Fitness:	0.5 * (hits/256) if hits < 256 0.5 + 0.5 * (17 – largest RPB depth)/(17) if hits = 256
Hits:	Same as Raw Fitness
Parameters:	M = 1000 (with overselection) G = 1001
Success Predicate:	None. Hits = 256 indicates a perfect adder, but does not terminate run.
Rules of Construction:	There is one RPB

The second approach uses the parameters in Table 2 for the evolution of the highest bit. The lower order bits are evolved separately and follow the same general parameters.

In the second approach the adjusted fitness is split in two equal ways. As before, one-half of the adjusted fitness is given to the correctness of the solution, and if the solution is perfect then the depth of the RPB (only one RPB this time) counts toward the fitness

Parameters are otherwise consistent with *Genetic Programming* (Koza 1992).

RPBs have equal chances of being chosen.
Crossover 90% (functions 90%, terminals 10%)
Reproduction 10%

In the interest of making efficient use of computer time, the runs for the second approach were initially run for only 201 generations. The most promising of the runs that evolved a perfect adder in this amount of time were then run for an additional 800 generations to allow the depth part of the fitness measure to exert selective pressure in the favor of faster perfect adders.

The evolutionary runs in this paper were implemented in LILGP, a genetic programming framework in C, and run on an Apple Powerbook G3/400MHz with 320MBs of RAM running MacOS X and on a Intel dual-Celeron 366Mhz machine with 128MBs of RAM running Mandrake Linux 8.1. The memory requirements for LILGP are minimal, most of the runs performed in this research consumed less than 10MBs of memory.

Results of the First Approach Applied to a 2-bit Adder

Figure 2: Generational Fitness of 2-bit Adder

```
current generation: 0
          generation: 0
               nodes: 79
               depth: 6
                hits: 38
         raw fitness: 38.0000
  standardized fitness: 27.0000
     adjusted fitness: 0.3958
RPB0:
 (or (xor (not b0)
          (not a0))
     (or (xor a0 a0)
         (xor a0 a0)))
RPB1:
 (not (not b1))
RPB2:
 (and b1 a1)
```

This run (Figure 2) starts out very characteristically with the mean fitness of the initial random population scoring 0.25 for blindly guessing and getting half of the output bits right. The best random individual (shown above) is somewhat better, scoring 38 of 48 hits. In generation 27 a perfect individual emerges with a depth of six. The depth portion of the fitness function becomes a driving factor and soon the best of run individual (shown below) emerges with a depth of 3.

```
=== BEST-OF-RUN ===
          generation: 64
               nodes: 33
               depth: 3
                hits: 48
         raw fitness: 48.0000
  standardized fitness: 3.0000
     adjusted fitness: 0.7059
RPB0:
 (or (xor (or a0 a0)
          (or b0 b0))
     (xor a0 b0))
RPB1:
 (xor (and a0 b0)
      (xor a1 b1))
RPB2:
 (or (or (and b1 a1)
         (and b1 a1))
     (and (and a0 b0)
          (xor a1 b1)))
```

Results of the First Approach Applied to a 4-bit Adder

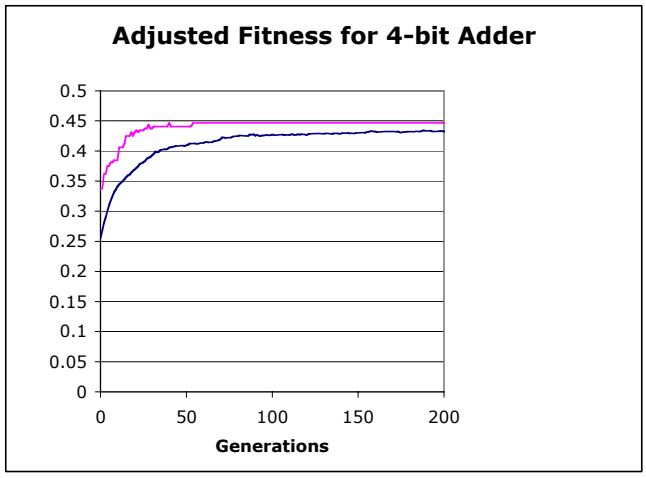

Figure 3: Generational Fitness of 4-bit Adder

This run (Figure 3) starts out typically with the mean fitness of the initial random population 0.25, but hits a plateau at generation 40 with a best, but flawed, solution (below) scoring 1144/1280 hits.

```
=== BEST-OF-RUN ===
     current generation: 201
           generation: 40
              nodes: 67
              depth: 8
               hits: 1144
          raw fitness: 1144.0000
   standardized fitness: 153.0000
       adjusted fitness: 0.4469
RPB0:
 (or (not (xor (and (or a0 a0)
                (and b0 a0))
            (or (not b0)
                (xor a0 a0))))
    (xor b0 a0))
RPB1:
 (xor (xor b1
         (and b0
            (not (not a0)))) a1)
RPB2:
 (xor (and (xor b2 a2)
         (xor b2 a2))
    (and b1
         (or (and (xor (and (xor b2 a2)
                            (xor (xor b1 b2) a2)) a0)
                  (and b1
                     (or (and b0 b1) a1))) a1)))
RPB3:
 (xor a2
    (xor b3 a3))
RPB4:
 (and b3 a3)
```

Results of the Second Approach Applied to a 4-bit Adder

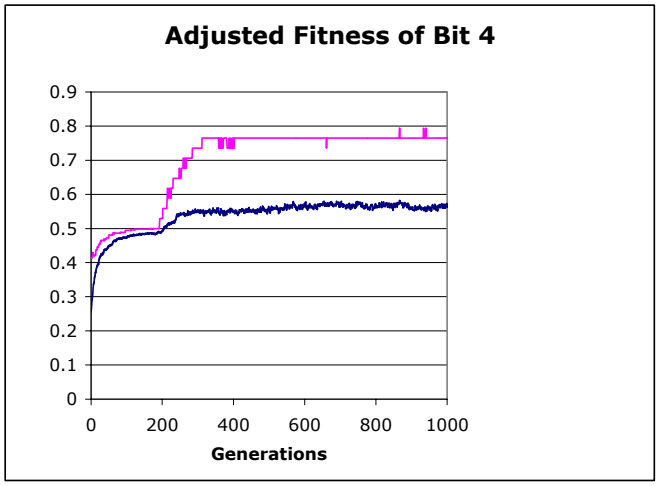

Figure 4: Generational Fitness of Bit 4

This run (Figure 4) begins with another typical start, and a perfect individual emerges at generation 168 with a depth of 17. This takes a hundred generations to shrink down to an individual of depth 8. Note the very small difference in fitness of the mean and the best just prior to the emergence of the first perfect individual. Below are the best of run individuals for each of the bits of the adder.

Bit 0:

generation: 0
nodes: 3
depth: 1
hits: 4
raw fitness: 4.0000
standardized fitness: 1.0000
adjusted fitness: 0.9706
RPB:
(xor b0 a0)

Bit 1:

generation: 45
nodes: 7
depth: 2
hits: 16
raw fitness: 16.0000
standardized fitness: 2.0000
adjusted fitness: 0.9412
RPB1:
(xor (and b0 a0)
(xor a1 b1))

Bit 2:

generation: 101
nodes: 19
depth: 4
hits: 64
raw fitness: 64.0000
standardized fitness: 4.0000
adjusted fitness: 0.8824
RPB:
(xor (not (not (and b1 a1)))
(xor (xor (not b2)
(not a2))
(and (and b0 a0)
(xor b1 a1))))

Bit 3:

generation: 811
nodes: 69
depth: 8
hits: 256
raw fitness: 256.0000
standardized fitness: 8.0000
adjusted fitness: 0.7647
RPB:
(xor (xor b3 a3)
(or (and (or (and (and a0 b0)
(and (and a1 b2)
(and a0 b0)))
(or b2 a2))

```
          (or (or (or (and a2 b2)
                      (and (and a0 b0) a1))
                  (and (or (and a0 b0)
                           (and a0 b0)) a1))
              (and b1
                   (or (and a0 b0) a1))))
      (xor (not (and (and (and a0 b0) a1)
                     (and a0 b2)))
           (not (and (and (and a0 b0) a1)
                     (or b2 a2))))))
```

Bit 4:

```
          generation: 866
             nodes: 115
             depth: 7
              hits: 256
         raw fitness: 256.0000
   standardized fitness: 7.0000
      adjusted fitness: 0.7941
RPB:
 (or (and (and (or (and (and a2 a2)
                        (or a3 b3))
                   (or (and (and b2 b0)
                            (and a0 b0))
                       (and a1 b2)))
              (or (and (and (or b1 a2)
                            (or a3 b3))
                       (or b2 b1))
                   (and (and a0
                             (or a3 b3))
                        (and a1 b0))))
          (or (and (and b0
                        (and (or a1 a1)
                             (and a2 a2)))
                   (and a0 b0))
              (or (or (and (and b2 b0)
                           (and a0 b0)) a1) b2)))
      (or (and (and (or (and a0
                             (or a3 b3))
                        (and (and a0 b0) a1))
                   (and (or (and a2 a2) a2)
                        (or a3 b3)))
              (and (and (and (or b2 b1)
                             (or b1 a2))
                        (or b0 b1))
                   (and (or b0
                            (and a0 b0))
                        (and a0 b0))))
          (and b3 a3)))
```

Discussion of the First Approach

In 18 runs using the first approach on a 2-bit adder, 5 perfect solutions evolved, all with a depth of 3 or 4. The first perfect adder from the run shown above was a complex solution with a depth of 9 that contained 165 nodes and over the generations gradually shrunk to the final result with a depth of 3 and containing 33 nodes. The unnecessary complexity of RPB0 is likely a failure of the third part of the adjusted fitness function to apply adequate pressure to favor generally faster solutions.

Applying the first approach to evolving a 4-bit adder evolved no correct solutions in several days of computer time. Increasing the population size, increasing the number of generations, and increasing the size of the individuals in the initial random population still did not evolve any perfect solutions. As one would expect the hardest cases for the adder to handle are 1111+0001, 1101+0011, and the like. In fact note that RPB3 and RPB4, for bits 3 and 4 respectively, fail to contain many of the necessary terminals to become correct solutions.

The evolution of the 4-bit adder failed for several reasons. The complexity of the problem is proportional to the bit-number (i.e. power of two) of the solution RPB. The time spent in the evaluation function is four times greater for each additional bit-size of the adder. A (n+1)-bit adder possesses two more inputs than an n-bit adder and thus has to cover quadruple the number of fitness cases. Indeed the RPBs for lower order bits found correct solutions sooner than higher order bits. This also means that the lower order RPBs were subjected to more fitness cases than necessary, consequently wasting computer-cycles.

Another significant obstacle to the development of the higher order RPBs is the fact that RPBs have equal probability of being chosen for crossover. Thus the lower order RPBs are chosen for crossover many more times than they need, and RPB3 and RPB4 are starved for the crossover operations they require to evolve better solutions.

From these failings the second approach was devised such that the tree of each resultant bit is evolved independently. In this manner the number of fitness cases over which each solution is evaluated can be kept to a minimum. And adequate selection pressure can be applied to each tree to evolve fast adder structures with minimum delay on all result bits.

Discussion of the Second Approach to a 4-bit Fast Adder

Although this approach took several days of computer time, it evolved a correct solution for a 4-bit adder, and spent less computer time on low order RPBs and more time on high order RPBs. Although the depth of 8 on bit 3 is not optimal it shrunk from an earlier solution of depth 17 and the solution is faster than cascading the 2-bit adders evolved earlier. Working with 2 input gates forces the solution to have greater depth than a solution working with commonly available 4 input gates.

Conclusion

This paper demonstrates two approaches to the automatic creation of a 2-bit time-optimal and a 4-bit near time-optimal digital fast adder circuit by means of genetic programming. It is important to stress that these results were obtained by running two older computers non-stop for one week. Of the two approaches taken the second approach proved more successful in the evolution of 4-bit fast adders.

Future Work

Normally outputs of digital circuits share some common logic. That is not the case in this research due to the approaches taken. A method that allows for sharing common logic should be beneficial to the evolution of larger circuits since the higher order outputs could borrow intermediate results from lower order circuits. The use of structural modifying operators on an embryonic circuit as described in (Koza et al, 1997) could facilitate the evolution of such shared logic.

Acknowledgements

I would like to thank Lockheed Martin Missiles and Space for their tuition grant. I would also like to thank Dr. Bill Punch and Dr. Erik Goodman for their work on LILGP.

References

J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992

John. R. Koza, Forrest H. Bennett, III, David Andre, Martin A. Keen, Frank Dunlap, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *IEEE Trans. Evolutionary Computation*, Vol. 1, No. 2, July 1997