Project 1

Write a C program that simulates cycle-by-cycle activity of an n-bit Carry Completion Adder

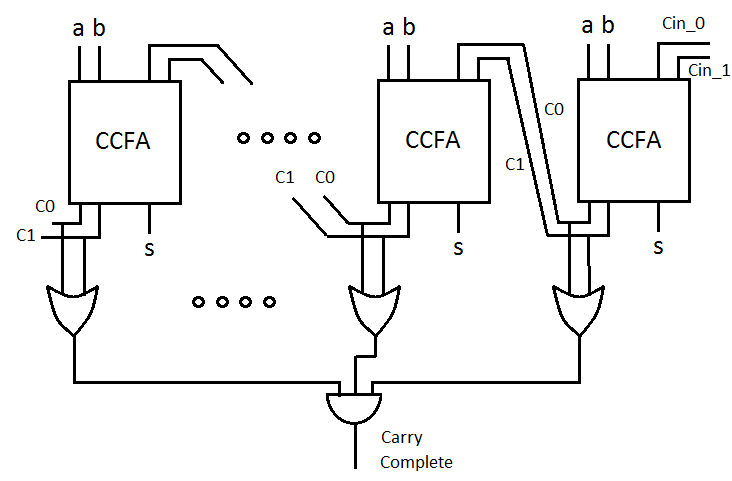
Robert A. Maxwell

EE 5423 Computer Arithmetic

October 15, 2013

The goal for this project is to write a C program that simulates an n-bit carry completion adder and to determine the relationship between average delay and operand size.

* The operand size varies from 1 to 48 bits, including the trivial 1-bit case.
* Performance of each set of operands is to be derived from the average of 1000 (1 million was actually used) sets of random input numbers.
* Simulation carries out the algorithm that is performed by hardware on a cycle-by-cycle basis.



The main sections included in this report are as follows:

1. The C program CCAdder-48bit-sim.c with documentation and optionally, the executable.
2. Simulation results and output of S, C0i and C1i at the end of each cycle given the predefined input patterns.
3. Plot of the average delay versus number of bits and include conclusions, reasoning.
4. Attached to this report is the code for the simulator in the C language. It was compiled and executed (binary executable is available upon request) in a flavor of Linux using the following commands:

gcc CCAdder-48bit-sim.c -o CCAdder

./CCAdder

2.) Here is the output from the simulator given the predefined input patterns listed below (screen shots are also available):

Input patterns:

A = 1000 0101 1100 1011 0010 0111

B = 0111 0110 1000 0100 1100 1001

[maxwell@Tesla Computer\_Arith]$ gcc CCAdder\_48bit\_sim.c -o CCAdder

[maxwell@Tesla Computer\_Arith]$ time ./CCAdder

Initially...

A = 000000000000000000000000

B = 000000000000000000000000

S = 000000000000000000000000

C0 = 000000000000000000000000

C1 = 000000000000000000000000

End of Cycle 1

A = 100001011100101100100111

B = 011101101000010011001001

S = 111100110100111111101110

C0 = 000010000011000000010000

C1 = 000001001000000000000001

End of Cycle 2

S = 111110010100111111100110

C0 = 000110000111000000110000

C1 = 000001011000000000000011

End of Cycle 3

S = 111111000100111111100100

C0 = 001110000111000001110000

C1 = 000001111000000000000111

End of Cycle 4

S = 111111000100111111110000

C0 = 011110000111000011110000

C1 = 000001111000000000001111

End of Cycle 5

S = 111111000100111111110000

C0 = 111110000111000111110000

C1 = 000001111000000000001111

End of Cycle 6

S = 111111000100111111110000

C0 = 111110000111001111110000

C1 = 000001111000000000001111

End of Cycle 7

S = 111111000100111111110000

C0 = 111110000111011111110000

C1 = 000001111000000000001111

End of Cycle 8

S = 111111000100111111110000

C0 = 111110000111111111110000

C1 = 000001111000000000001111

End of Cycle 9

S = 111111000100111111110000

C0 = 111110000111111111110000

C1 = 000001111000000000001111

Number of bits: 24

Average Delay = 18.000

real 0m0.004s

user 0m0.002s

sys 0m0.001s

1. As seen in the figure below, there is a minimum 4d or 2 cycle delay for this hardware in the 1-bit case. The curve of the delay increases with the number of bits but begins to saturate at bit sizes greater than 48. This is due to the fact that one of the carry-out signals may be known in one cycle if Ai = Bi = 1 or 0.

C0 = ai AND bi

C1 = ai’ AND bi’

The previous carry chain is effectively halted when Ai and Bi hold either a (0,0) or (1,1) which starts a new carry chain. There is an equal probability (0.5) of having either a (0,0) ,(1,1) or (1,0),(0,1) at the inputs. Likewise there is a very low probability of the worst case :

A = 10101010101

B = 01010101011

P(X = 1,1 or 0,0) = ¼ + ¼ = ½

P(X = 0,1 or 1,0) = ¼ + ¼ = ½

But, the probability that (0,1) is followed by a (0,1) or (1,0) is ½ \* ½ = 1/4. This only gets worse (½)n as the number of bits increases: 1/8 for 3-bits, 1/16 for 4-bits, etc. So as the number of bits grows, there is a 50% chance that both bits to be added will be either (0,0) or (1,1) which effectively stops to propagation of the carry chain.

Upon inspection of the graph, the curve follows a known exponential model. In the following figure, the model a + b\*ln(x) (x = number of bits), is overlaid on the data taken from the simulator. There is a good match between the model and the simulator data.