Report
On

Design and Timing analysis of Dadda Multiplier

Apoorv Dethe

Date: Dec’18

Problem statement

Assume the following delay values for half and full adders:
Half adder: carry (AND gate) = 40 ps, sum (XOR) = 70 ps.
Full adder: carry (AB + BC + CA) = 80 ps, sum (A XOR B XOR C) = 120 ps.

Describe and simulate a Dadda 8x8 Multiply and Accumulate circuit in verilog, (Icarus verilog – a public domain verilog simulator will be used for this assignment).

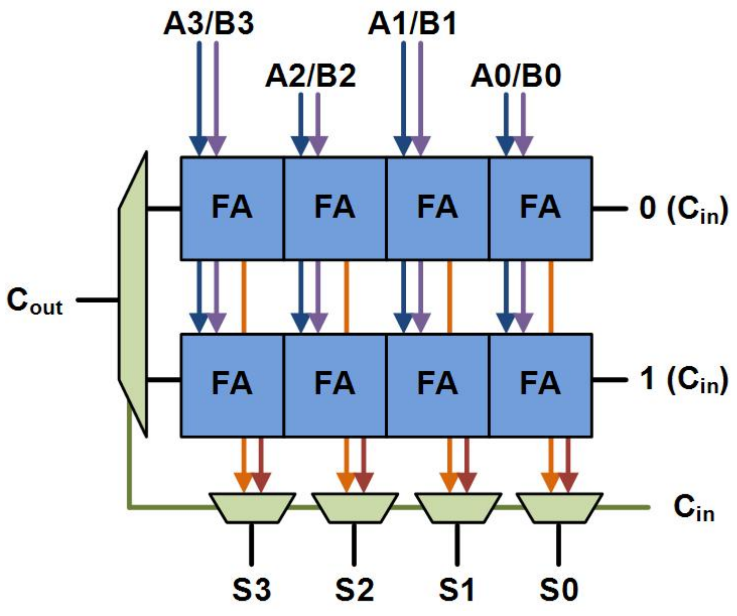
Half adders and full adders can be used as behaviourally described building blocks with delays as specified above. The accumulator should be 16 bits wide. Use a carry select adder for the final addition, assuming the carry select mux delay to be 50 ps. A flag output indicating overflow in the accumulator register should be provided.

Notice that the sum and carry delays are not equal. Make design choices for allocation of wires to specific adders and sizes of sub adders in the final carry select adder to reduce the overall worst case delay.

Simulate the design with 16 random test vectors for multiplicand, multiplier and initial content of the accumulator register. Also simulate for multiplication of FF by FF with the initial value of the accumulator being 0F0F

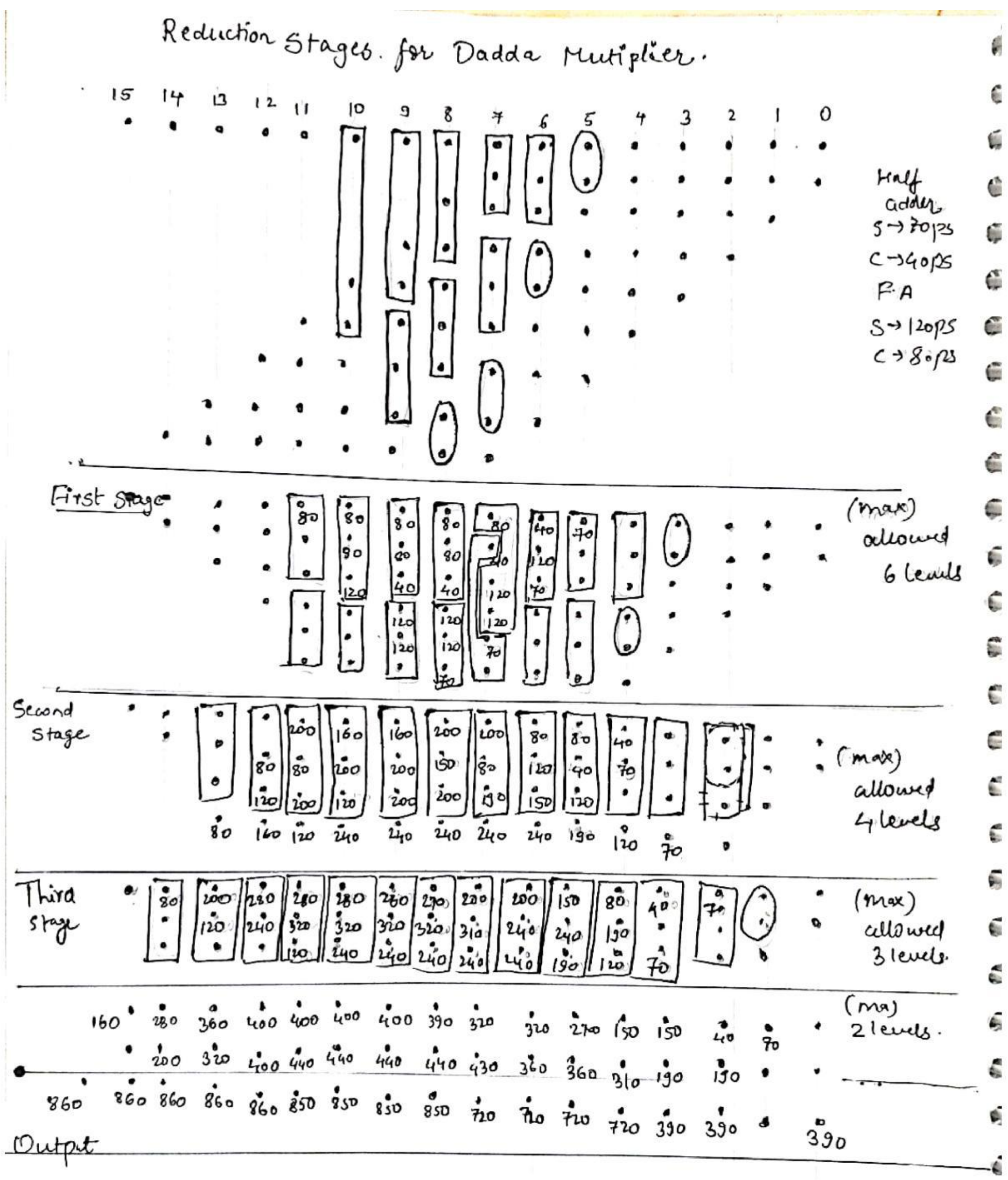
Dadda Multiplier

Dadda multipler is a hardware multiplier design which reduces number of partial products by stages of half adders and full adders until we are left with at most two bits of each weight. Final result is added with conventional adder.
In our problem statement we are using carry select adder as the final adder. The block diagram of carry select adder is as shown below.



Carry Select adder

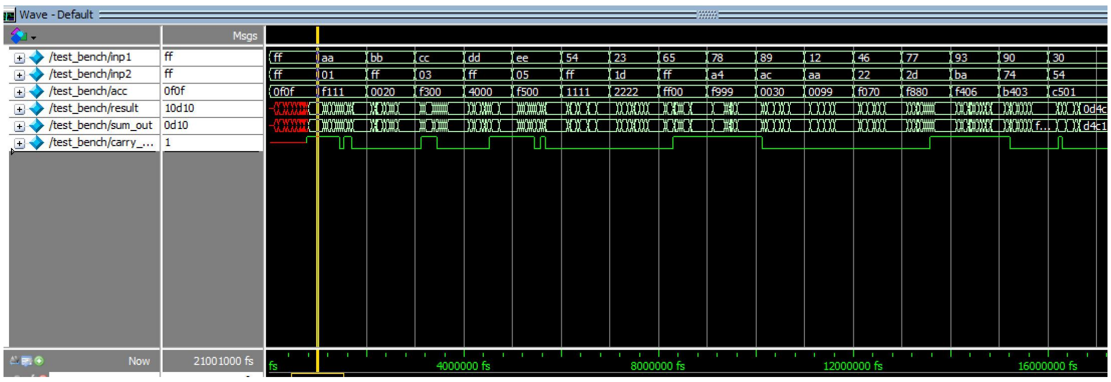
The following diagrams shows the reduction stages for Dadda multiplier. The delays for sum and carry for half adder and full adder is given in problem statement.



Reduction diagram

The code for Dadda multiplier is written in verilog and simulated in Modelsim.

The simulated outputs for 16 random test vectors is shown and also the output for FF*FF with accumulator contents 0F0F is also shown.



```
/test_bench/inp1     ff     ff    aa    bb    cc    dd    ee    54    23    65    78    89    12    46    77    93    90    30
/test_bench/inp2     ff     ff    01    ff    03    ff    05    ff    1d    ff    a4    ac    aa    22    2d    ba    74    54
/test_bench/acc      0f0f   0f0f  f111  0020  f300  4000  f500  1111  2222  ff00  f999  0030  0099  f070  f880  f406  b403  c501
/test_bench/result   10d10  0d4c
/test_bench/sum_out  0d10   f...  d4c1
/test_bench/carry_…  1
```

Output waveform