Designing multiplier blocks with low logic depth.

Andrew Dempster
Suleyman Demirsoy
Izzet Kale

Cavendish School of Computer Science

Designing Multiplier Blocks with Low Logic Depth

A G Dempster, S S Dimirsoy, I Kale

University of Westminster, 115 New Cavendish St, London W1W 6UW

Tel: +44 20 7911 5891, Fax: +44 20 7580 4319, email: dempsta@cmsa.wmin.ac.uk

Abstract - The depth of logic in an integrated circuit, particularly a CMOS circuit, is highly correlated both with power consumption and degraded switching speed. Hence, designs with low logic depth can aid in reducing power consumption and increasing switching speed. In this paper we demonstrate how new and modified algorithms have been used to design multiplier blocks with low logic depth and power consumption.

1.0 INTRODUCTION

1.1 Background

Low-power circuit techniques are becoming ever more important. Power consumption of a node in a synchronous CMOS circuit can be estimated using:

$$P_{switching} = \alpha C_L V_{dd}^2 f_{clk} \qquad (1)$$

where $P_{switching}$ is the power consumption due to switching, α is the probability of a 0 to 1 or 1 to 0 transition in a clock period, C_L is the load capacitance, V_{dd} is the supply voltage and f_{clk} is the switching clock frequency. Reducing the logic depth of a circuit will reduce the length of paths via which glitches can propagate, thereby reducing α. Our early results show that logic depth is a useful high-level indicator of relative power consumption, i.e. of two structures, the one with the higher logic depth generally has higher power consumption [1].

Multiplier blocks are structures made up of connected two-input adders, configured to produce products of an input multiplicand and one or more coefficients. They are highly efficient replacements for dedicated multipliers, where fixed-point constant coefficients are required. Multiplier block algorithms [2]-[5] have always aimed at minimising the number of adders required to perform one [3][4] or more [2][5] multiplications of a single multiplicand. The latter is useful in FIR filtering, IIR filters [6][7] and filter banks [8]. All these algorithms exploit redundancy in the shift-and-add multiplication process. The output of the algorithms is a graph with vertices representing adders and edges representing shifts. The output of any adder is labelled with an odd integer (called a *fundamental*) which represents the (possibly partial) product at that point in the graph. The cost was conventionally measured only in "adders" (which also includes subtractors) as shifts can be performed by wiring and are thus "free". In general fewer components implies lower power, but this is not always the case [9]. Long paths in the network can allow glitches to propagate via many adders , i.e. structures having high *logic depth* consume more power.

We present here some of the key results of a survey of multiplier block design algorithms [10].

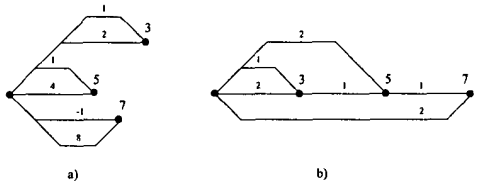

Figure 1 Multiplier blocks producing multipliers 3, 5 and 7 using three adders: a) logic depth 1, b) logic depth 3. The only fundamentals in both graphs are 3, 5, and 7.

2.0 MINIMUM-DEPTH GRAPHS

When processing a set of coefficients, a series of decisions as to how to connect new adders into the graph are made which are critical to the eventual logic depth. We use the set (3, 5, 7) to illustrate these "connection decisions". We start by creating {3 = 2 + 1} (or possibly {3 = 4 - 1}), a graph of logic depth 1. With 3 already in the graph, we either synthesise 5 as {5 = 3 + 2 = say (2+1) + 2}, which has logic depth 2, or {5 = 4 + 1} which has depth 1. Similarly, adding 7 to the graph could be achieved as {7 = 5 +2} or {7 = 2*5 - 3}, with depth 2 or 3, {7 = 3 + 4} or {7 = 2*3 +1}, with depth 2, and {7 = 8 - 1}, with depth 1. Minimum-depth decisions lead to a depth-1 graph; maximum-depth decisions lead to a depth-3 graph, as shown in Figure 1. The poorly-performing example in [9] had logic depth 2 for these three coefficients. Note that all of the above combinations are optimal in terms of fewest (3) adders. Processing a set of fundamentals in the order they were added to the graph by a design algorithm, using minimum-depth decisions we call Minimum-Depth Processing (MDP).

0-7803-7448-7/02/$17.00 ©2002 IEEE

3.0 SINGLE-COEFFICENT ALGORITHMS

We tested three existing algorithms: MAG [3], an exhaustive search of graph topologies, BHM [5], derived from the Bull and Horrocks algorithm [2] and BERN [4], based on Bernstein's algorithm [11].

We also defined a new algorithm, the MAGL algorithm, which makes use of MDP:

i) Use the outputs of the exhaustive MAG algorithm, i.e. the table containing the adder costs the table containing all fundamentals that can be used to synthesise each coefficient.

ii) For each candidate graph, use MDP and evaluate its logic depth.

iii) Select all graphs of minimum depth.

iv) From these, select the graph with the lowest sum of fundamentals (thereby reducing data wordlength which should also help reduce power)

Whereas MAG produced a set of graphs for each coefficient, MAGL selects a particular graph.

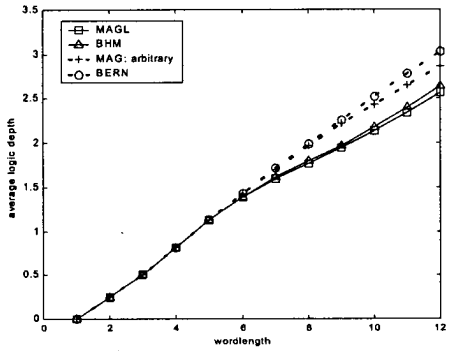

Figure 2 The logic depth for single multipliers designed by the MAGL, BHM and BERN algorithms (averaged over all coefficients of a given wordlength). Also shown is a representative arbitrarily (equiprobable) selected from all possible MAG graphs.

Figure 2 compares the four algorithms. MAG is represented by an arbitrary (equiprobable) selection from its set of possible graphs. MAGL always performs best graph. BHM performs very well. Interestingly, the arbitrary MAG choice performs badly. This is because shallow graphs are rare, e.g. of the 7 cost-3 graph topologies, only one has depth 2 (see Figure 3). Graph 7 in Figure 3 has lower depth because it makes use of parallelism, or a tree structure [12]. BERN never uses parallelism, which is why it performs poorly.

For wordlengths longer than 12 bits, MAG and MAGL cannot be used, due to computational limitations. For wordlengths up to 32 bits, BHM was found to perform very well and produces designs with low logic depth [10].

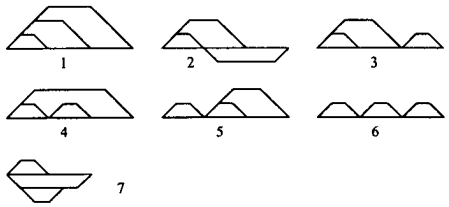

Figure 3 The seven three-adder graph topologies for single multipliers. Graphs 1-6 have logic depth 3; graph 7 has logic depth 2 [3].

4.0 MULTIPLE-COEFFICIENT ALGORITHMS

4.1 Short Wordlengths

When designing multiplier blocks to replace several coefficient multipliers, the algorithms that produce designs with fewest adders are RAG-n [5] for short wordlengths (it is limited by its use of the MAG tables) and BHM [5] for long wordlengths.

Results (dotted lines) in Figure 4 show that for between 10 and 40 coefficients, BHM is superior to RAG-n. For more than 30 coefficients, the logic depth of RAG-n decreases as set size increases, due to the RAG-n "cost-1 problem". RAG-n initially places into the graph any cost-1 coefficients (i.e. 3, 5, 7, 9, 15, 17 etc.). If one of the required coefficients is cost-1, the algorithm is more likely to complete the design without resorting to heuristics. As the probability of a cost-1 in the set increases, the resulting graph is more likely to be optimal (i.e. it has fewest possible adders) [5]. As set size increases, the likelihood of a cost-1 coefficient (and hence optimality) increases, as also shown in Figure 4. This has an effect on logic depth also because there seems to be a close relationship with the decline in logic depth and the incidence of optimality.

For both RAG-n and BHM, the resulting graphs were fed back into the original algorithms, i.e. both algorithms were applied twice (solid lines in Figure 4). The reasoning for this is that when building up a difficult graph, both algorithms will add cost-1 fundamentals that are not in the original coefficient set. Both algorithms process cost-1 coefficients first and placing cost-1 fundamentals in the graph early tends to produce graphs with shorter depth. The peak in logic depth for RAG-n is much reduced,

because there are now guaranteed cost-1 coefficients to work with. However, as the likelihood of optimality increases (the set size is increased), the benefit of running the algorithm twice diminishes. In general, RAG-n performs better when run twice than any other algorithm.

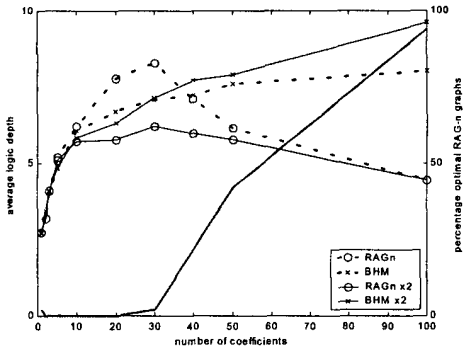

Figure 4 Logic depth for sets of up to 100 12-bit coefficients, averaged over 50 sets at each set size. RAG-n and BHM are both applied to the original coefficient set and then a second time to the resulting fundamentals. The bold line, against the right axis, shows the percentage of RAG-n graphs known to be optimal.

So RAG-n remains a superior algorithm to BHM, but only if the following procedure is followed:

i) design the multiplier block using RAG-n

ii) re-design the graph using RAG-n applied to the fundamentals of the first graph

4.2 Long Wordlengths

The experiment illustrated in Figure 5 examines the performance of BHM with respect to MDP, arbitrary decision making, and maximum-depth decisions. We can see that BHM does not design minimum-depth graphs, but they are better than if purely arbitrary decisions were made.

4.3 The C1 Algorithm

Having established that cost-1 fundamentals help reduce logic depth, we designed the C1 algorithm:

i) Use RAG-n (or BHM for long wordlengths) to design a multiplier block, using all the required coefficients plus all cost-1 coefficients up to twice the value of the maximum coefficient. The logic depth of this graph is the "target depth"

ii) Eliminate from this graph all cost-1 coefficients not used to create any of the required coefficients. We now have a "useful" set of cost-1 coefficients. The cost of this graph is the "current cost"

iii) For each coefficient in the useful set, starting with the largest (i.e. least likely to be useful), test if RAG-n can design a graph without that coefficient, costing one less than the current cost but not increasing logic depth. If so, eliminate it from the useful set. Decrement the current cost and try the next coefficient.

The algorithm is quite computationally intensive.

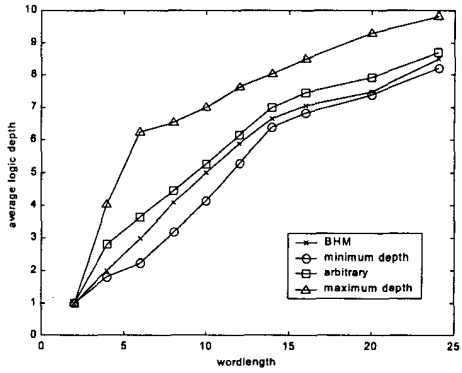

Figure 5 Comparison of average logic depth for 50 sets of 10 coefficients of different wordlengths. BHM is compared with post-processing the BHM fundamentals producing minimum- and maximum-depth graphs, and making arbitrary choices.

An experiment comparing C1 with RAG-n and RAG-n run twice is shown in Figure 6. Again, applying RAG-n twice produces shallower graphs, but C1 is better. A small average adder cost penalty (2.5% for 12-bit coefficients) is incurred by C1.

4.4 An Example FIR Filter

An arbitrary FIR filter specification (Remez, with normalised $f_p = 0.25$, $f_s = 0.3$, equal ripples in pass- and stop-bands, order 24) gave "floored" 12-bit coefficients {-710, 327, 505, 582, 398, -35, -499, -662, -266, 699, 1943, 2987, 3395, 2987, 1943, 699, -266, -662, -499, -35, 398, 582, 505, 327, -710}. Results are in Table 1. BHM produces a shallower result than RAG-n in this example, despite using more adders. Reapplying the algorithms doesn't help. Applying the C1 algorithm, although costing one more adder than RAG-n, drastically reduces the logic depth.

Each design was synthesised using Leonardo for a Xilinx Virtex 300 BG432-4 and simulated using Modelsim back-annotated simulation with 1 ps precision. Arithmetic was 2's complement and the input was 512 uniformly distributed inputs in (-128, 127). Transition counts are also shown in Table 1.

The simulations support the idea that logic depth is a good indicator of power consumption. RAG-n designed the filter with highest power consumption, despite having fewest adders. As expected, C1 designed the best (most power efficient) filter.

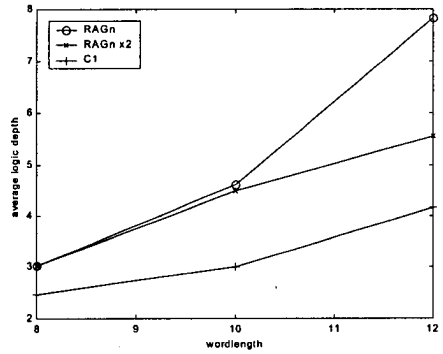

Figure 6 Average logic depth for short wordlengths over 50 sets of 25 coefficients: comparing the C1 algorithm with RAG-n applied once and twice. RAG-n results are identical to RAG-n

Design	Adder Cost	Logic Depth	Transitions
RAG-n	18	9	1766134
BHM	20	5 (5)	1037407
RAG-n x2	18	9	
BHM x2	20	5 (5)	
C1	19	4	944440

Table 1 Adder cost, logic depth and transition count for the example filter multiplier block for RAG-n and BHM, applied once and twice, and C1. For BHM, the result of MDP is shown in brackets.

6.0 CONCLUSION

We introduce the following methods which we have shown to reduce logic depth in multiplier blocks:

i) Minimum-depth processing of a set of fundamentals,

ii) The MAGL algorithm which selects the shallowest of the possible MAG graphs,

iii) Applying RAG-n, then re-applying it to the fundamentals produced by the first application, and

iv) The C1 algorithm.

Logic depth is shown to be a good measure of power consumption, better than adder cost, and C1 is shown to design an efficient filter.

7.0 REFERENCES

[1] S S Demirsoy, A G Dempster and I Kale, "Transition analysis in multiplier-block based FIR filter structures", submitted to ICECS2000

[2] D R Bull and D H Horrocks, "Primitive operator digital filters", IEE Proceedings G, vol 138, no 3, pp401-412, Jun 1991

[3] A G Dempster and M D Macleod, "Constant integer multiplication using minimum adders", IEE Proceedings - Circuits, Devices and Systems, vol 141, no 5, pp407-413, Oct 1994

[4] A G Dempster and M D Macleod, "General algorithms for reduced-adder integer multiplier design", Electronics Letters, vol 31, no 21, pp1800-1802, Oct 1995

[5] A G Dempster and M D Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters" IEEE Trans Circuits and Systems II, vol 42, no 9, pp569-577, September 1995

[6] A G Dempster and M D Macleod, "IIR Digital Filter Design Using Minimum-adder Multiplier Blocks", IEEE Trans Circuits & Systems II - Digital & Analog Signal Processing, vol. 45, no. 6, pp. 761-763, June 1998.

[7] A G Dempster, "The Cost of Limit-Cycle Elimination in IIR Digital Filters Using Multiplier Blocks", Proc. ISCAS97, vol 4, pp 2204-2207, June 1997

[8] A G Dempster and N P Murphy, "Efficient Interpolators and Filter Banks using Multiplier Blocks", IEEE Trans Signal Processing, vol 48 no 1, pp 257-261, Jan 2000

[9] David H Horrocks and Yodchai Wongsuwan, "Reduced Complexity Primitive Operator FIR Filters for Low Power Dissipation", Proc ECCTD '99, Stresa, Italy, pp273-276, 1999

[10] A G Dempster, "Algorithms for Reducing Logic Depth in Multiplier Blocks", submitted to IEEE Trans C&S II

[11] Robert Bernstein, "Multiplication by Integer Constants", Software-Practice and Experience, vol 16, no 7, pp641-652, Jul 1986

[12] N G Kingsbury, "High-speed binary multiplier", Electronics Letters, vol 7 no 10, pp 277-278, 1971