Deriving Optimal Multiplication-by-Constant Circuits With A SAT-based Constraint Engine

Vitaly Lagoon
Cadence Design Systems, USA
lagoon@cadence.com

July 22, 2020

Abstract

Constant multiplication circuits can be realized by using additions, subtractions and left-shifts. The problem of finding a multiplication circuit with minimum number of adders and subtracters for a given constant (set of constants) is known as single (multiple) constant multiplication (SCM, MCM) problem.

In this work we demonstrate how SCM and MCM instances can be encoded in constraints of SystemVerilog language and solved by the integrated constraint solver of Xcelium Logic Simulator by Cadence. Given a target constant C and a number N of nodes (adders and subtracters), our approach produces a description of a circuit of N nodes taking an arbitrary input x and computing Cx. The technique is *complete*, proving non-existence of a circuit if the corresponding multiplication requires more than N nodes. Thus, an *optimal* realization can be derived for Cx by searching for the lowest N for which the realization is feasible.

Our experiments show that the integrated constraint-solving flow of Xcelium based on a *Boolean satisfiability solver* can handle hard and large instances of SCM/MCM, such as multiplication by high 32-bit constants in reasonable time, usually no more than a few seconds.

We conjecture that the approach can be relevant in the context of IC/FPGA realization.

1 Introduction

Multiplication by constant is a performance-critical element of many numeric algorithms in *digital signal processing* (DSP) and control applications. One well-known approach to speeding-up a constant multiplication $y = Cx$ for a known integer or fixed-point constant C in the contexts of ASIC and FPGA realizations is to synthesize a specialized (with respect to C) multiplying circuit based on adders, subtracters and left-shift operations. The problem of finding such circuits with minimum numbers of adders and subtracters is known as *single constant multiplication* or SCM. The generalization of the problem for multiplication by a set of given constants is known as as *multiple constant multiplication* or MCM. It is shown that SCM and MCM, by trivial generalization, are NP-hard [6].

There is a straightforward upper bound of $N = \lceil \log_2(C)/3 \rceil$ nodes for a multiplication by C, based on the *canonical signed digit* (CSD) encoding [5]. The state-of-the-art literature reports much tighter bounds for N, and conjectures that the minimum cost is sub-linear with respect to the size of binary encoding of C (or sub-logarithmic with respect to C). The exact minimum cost of SCM circuits however, remains an open research problem [12].

Numerous techniques of finding minimal or nearly minimal circuits for constant multiplication (SCM and MCM) have been reported in the literature. The survey of the methods is beyond the scope of this short paper. The reader is referred to [10, 7, 8] for more details. The methods proposed in the literature are based on encoding optimizations, graph reduction techniques and more recently, on applications of Integer Linear Programming (ILP). In general, the proposed methods are divided into *precise* and *approximate*. Precise methods target exact minima of adders and

subtracters while the approximate techniques usually apply sets of heuristics leading to reduction in circuit sizes, but not guaranteed to find the global minima.

In this work we demonstrate a precise method of solving SCM/MCM by modeling the corresponding multiplication circuits in a mix of arithmetic and bitwise constraints over finite bit-vectors, and deriving the realization circuits by solving the constraints in a SAT (Boolean satisfiability) engine. We use the constraint subset of SystemVerilog [2, 3] verification language for our models. However, we claim that the model is generic and should be easily portable to any common constraint language supporting bit-vectors. We conduct several experiments based on solving the models for several SCM instances of interest. The solving is done using Cadence® Xcelium™ Parallel Logic Simulator for SystemVerilog [11] which integrates several constraint solving engines. In all our experiments the actual solving is done by the SAT flow of the Xcelium Simulator. That flow incorporates several layers of program analysis, simplification, and translation of bit-vector constraints to SAT. The actual SAT solving is performed by a variant of Glucose SAT solver [1] modified for finding randomized solutions of the corresponding *conjunctive normal form* (CNF).

Unlike most of prior art on SCM/MCM this paper does not propose any special-purpose algorithms for solving the optimal multiplication problem. Instead, we rely on the existing general-purpose Xcelium™ engine and its integrated SAT solver. The contributions of the paper are thus, (a) presenting a clear and concise model for the problem that can be straightforwardly translated to SAT and (b) demonstrating through the series of experiments that the method is practical.

The closest to our approach is perhaps the work of Aksoy *et al.* [4]. In that work, elaborate boolean modeling is used for capturing sharing of sub-terms, and a 0-1 ILP solver based on SAT is used for finding solutions with maximum sharing which reflect the minimum of required adders and subtracters. We believe that the model presented in this work is more intuitive and natural by comparison, which makes it easier to understand and adopt in practical contexts. Moreover, since we model the SCM/MCM problem directly in terms of the adders, subtracters and shift operations of the circuit, the corresponding correctness and optimality of the approach follow trivially, with no need for proofs.

2 The Constraint Model of SCM Circuits

The proposed method is based on constraint encoding of the SCM problem in the embedded constraint language of SystemVerilog [2, 3]. The (slightly simplified) model for SCM is shown in Figure 1. We model a linearized view of a multiplication circuit as an array `scm.nodes[`N`]`. Each element in the array represents an adder or a subtracter. The nodes relate to their input arguments through the indices `i1` and `i2` in the array of nodes. The constraint in lines 30–34 equates the input variables of each node `v1` and `v2` with the output values of the argument nodes designated by `i1` and `i2`. The constraints in lines 13–17 establish the straightforward relations between variables of each node. The `target` constraint in line 35 requires that the last node in the linearized sequence outputs the target multiplication result.

The model of Figure 1 produces randomized solutions for the case defined by the corresponding ‘N and ‘C constant as illustrated by the following example.

Example 1. Consider the problem of optimal multiplication by 12345. Figure 2 demonstrates the output (left) and the math formula (right) of the corresponding circuit produced by the constraint solver for the encoding of Figure 1. It takes less than 0.5sec to derive the solution. It is also easy to prove based on the same model that the result is optimal. By running the same model with the definition “`‘define N 3`” we get a contradiction in less than 0.5sec. The infeasibility result means the same multiplication cannot be handled by less than four adders/subtracters.

The results of the above example demonstrate that the proposed method is useful not only for deriving multiplication circuits, but is also capable of proving their minimality. The example suggests a natural method of optimizing SCM instances. First, we find a sub-optimal solution based on a conservative estimate of N. The most conservative estimate for which a solution always exists is $N = \lceil \log_2(C)/3 \rceil$, but better (lower) estimates are not hard to guess (see Table 1 below

```
00  `define N 4
01  `define C 12345;
02
03  module top;
04  class node;
05     rand bit ADD;                  // is this an ADD or a SUB node?
06     rand int unsigned i1,  i2;     // the indices of the two input nodes (or 0 for x)
07     rand int unsigned v1,  v2;     // values copied from the input nodes
08     rand int unsigned sh1, sh2;    // left-shift the inputs by that many bits
09     rand int unsigned o;           // the output of the node
10     rand int unsigned v1sh, v2sh;  // intermediate vars
11
12     constraint node_rels {
13        i1==0 -> v1==1;                      // if i1 refers to the input of the circuit
14        i2==0 -> v2==1;                      // if i2 refers to the input of the circuit
15        v1sh==v1<<sh1; v1==v1sh>>sh1;        // avoid overflow when shifting v1
16        v2sh==v2<<sh2; v2==v2sh>>sh2;        // avoid overflow when shifting v2
17        (ADD) ? o==v1sh+v2sh : o==v1sh-v2sh;  // compute the output of the node
18     }
19  endclass // node
20
21  class scm;
22     int unsigned max_shl;        // maximum shift
23     int unsigned val;            // target values
24     rand node nodes[];           // linearaized circuit
25
26     constraint init  { nodes[0].v1==1; nodes[0].v2==1; }
27     constraint limits { foreach (nodes[i]) { nodes[i].sh1<max_shl; nodes[i].sh2<max_shl;
28                                              nodes[i].i1<=i;       nodes[i].i2<=i; }}
29
30     constraint val_mux {
31       foreach(nodes[i]) foreach (nodes[j]) if (j<i) {
32         (nodes[i].i1==j+1) -> (nodes[i].v1==nodes[j].o);
33         (nodes[i].i2==j+1) -> (nodes[i].v2==nodes[j].o);
34     }}
35     constraint target { nodes[`N-1].o==val; }
36  endclass // scm
37
38     scm SCM = new;
39     initial begin
40        SCM.nodes = new[`N];
41        foreach (SCM.nodes[i]) SCM.nodes[i] = new;
42        SCM.val = `C;
43        SCM.max_shl = $clog2(SCM.val)+1;
44        SCM.randomize();
45        // display SCM (omitted)
46     end
47  endmodule
```

Figure 1: The basic SCM model

`N1 = (X<<3) - X`	$8x - x$	$= 7x$
`N2 = (X<<5) - N1`	$32x - 7x$	$= 25x$
`N3 = (N1<<6) + N1`	$64 \cdot 7x + 7x$	$= 455x$
`N4 = (N2<<9) - N3`	$512 \cdot 25x - 455x$	$= 12345x$

Figure 2: Multiplication by 12345 as a 4-node circuit

and the corresponding explanations). Once the initial value of N is established, the minimization consists in solving the same model with decreasing values of N. The lowest value for which the solution exists is the proven minimum of nodes required for the given SCM instance.

For our next experiment we used a slightly modified version of the original model. We changed several elements of the encoding, for example we used *unary* encoding of `i1` and `i2`, as it facilitates faster SAT solving. We also added several *symmetry breaking constraints* that prevent the solver from inspecting symmetric solution candidates and thus, reduce the search space.

In this experiment we verify the known lower limits for constant multiplications that cannot be realized in a given number of nodes, from one to six inclusively. Table 1 presents the lowest constants that cannot be realized in the corresponding number of N adders and subtracters. To the best of our knowledge, the constants for $N > 6$ are unknown, and the value for $N = 6$ has only been conjectured i.e., it is not known if there is a lower value for which the multiplier cannot be realized in six adders and subtracters [9].

For each entry we show the times (in seconds) it took to prove that the corresponding instances are unsatisfiable. For $N = 1, 2$ and 3 the results are instantaneous. In the cases $N = 4, 5$ and 6 we ran a parallel Glucose SAT solver with 20 threads on a machine with 28-core Intel Xeon 2.6GHz CPU's. The "Real Time" column shows the time it took to get the unsatisfiability result in each case. The "CPU time" shows the cumulative CPU time spent by all the 20 threads[1]. As the reader can see, it took us over 7 hours of real time and the equivalent of 6 days of machine time to confirm the result for $N = 6$. Again, to the best of our knowledge, it is the first time it was proven[2] that multiplication by 171,398,453 cannot be realized in six adders and subtracters.

N	C	Real Time	CPU Time
1	11	0	0
2	43	0	0
3	683	0	0
4	14,709	0.9	15.1
5	699,829	106	2,111
6	171,398,453	26,150	520,751

Table 1: Verifying the minimum values that cannot be realized in N nodes

Please note however, that in this example we consider very hard instances of the problem, where no realization exists and the solver is required to prove it. The computation is much shorter in cases when the realization exists, as illustrated by the following example.

`N1 = (X<<7) + X`	$2^7x + x$	$= 129x$
`N2 = N1 - (X<<4)`	$129x - 2^4x$	$= 113x$
`N3 = (N2<<4) + X`	$2^4 \cdot 113x + x$	$= 1809x$
`N4 = (N2<<19) + N3`	$2^{19} \cdot 113x + 1809x$	$= 59246353x$
`N5 = N4 - (N1<<14)`	$59246353x - 2^{14} \cdot 129x$	$= 57132817x$
`N6 = (N5<<2) - N5`	$2^2 \cdot 57132817x - 57132817x$	$= 171398451x$

Figure 3: Multiplication by $171,398,451$ as a 6-node circuit

Example 2. Consider the circuit realization $171,398,451x$. The multiplier is an odd number closest from below to the value for $N = 6$ in Table 1. (Even multipliers are easier. We can solve the problem for $C/2$ and left-shift the output by one.) The 6-node multiplier for that SCM instance is shown in Figure 3. It took the parallel Glucose solver running 20 threads only 7.5

[1]It is interesting to note that we gain nearly $\times 19$ speed-up in $N = 5$ and $N = 6$ cases by employing 20 threads. Obviously, parallel Glucose scales very well, at least for the given problem.

[2]Albeit, through "mechanical" means of theorem proving

seconds to derive this circuit description. Proving that the same multiplication cannot be realized in five nodes takes 56 seconds.

MCM In this short paper we do not go into details about solving *Multiple Constant Multiplication* (MCM) problem. Let us only state that the same approach extends naturally to accommodate MCM. In case of SCM we constrain the result computed in the final step to be the target value. For MCM we generalize that requirement specifying that the set of target values is included in the set of results computed by the nodes of the circuit. Our initial experiments indicate that the method can solve MCM instances faster than the state of the art methods reported in the literature.

3 Future Work And Conclusion

There are several directions in which this work can continue and develop.

1. We have already identified several extensions and refinements of the basic model of Figure 1 which reduce the CPU and memory resources required for the computation. We need to experiment with additional symmetry breaking and other improvements of the encoding. Adding symmetry-breaking cen be beneficial in proving optimality bounds, when we need to find a value of N for which the model is unsatisfiable.

2. In addition to the standard 2-argument adders some modern FPGA hardware implements *ternary adders* that compute sums of three inputs in one hardware unit. Our proposed method can be extended to allow for ternary adders in SCM/MCM solutions, thus further reducing the node counts in the multiplier circuits.

3. There exist several metrics which provide indirect measures of power consumption of shift-and-add based constant multiplication circuits. The simplest one is the *adder depth* (AD). It is defined as a maximum number of cascaded adders on each path. The *glitch path count* (GPC) was introduced as a more accurate power estimation. Without going into the details let us state that the proposed method naturally extends for adding AD or GPC as secondary minimization objectives. Once the minimum number of nodes is determined for an SCM (or MCM) instance, we can find the best solution in terms of the chosen power estimation.

To conclude, we have demonstrated how a problem of finding optimal multipliers by constant (SCM) can be encoded in SystemVerilog constraints and solved by Cadence® Xcelium™. We demonstrate that the method can handle non-trivial SCM instances in reasonable times. We also make a theoretical contribution by confirming for the first time that multiplication by 171,398,453 cannot be realized in six adders/subtracters. We argue that the proposed approach naturally extends in several directions for related tasks such as MCM, and for more optimizations of the target circuits.

References

[1] The Glucose SAT solver. `http://www.labri.fr/perso/lsimon/glucose/`.

[2] IEEE standard for SystemVerilog–unified hardware design, specification, and verification language. *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, pages 1–1315, Feb 2018.

[3] SystemVerilog — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/SystemVerilog`, 2018.

[4] Levent Aksoy, Eduardo A. C. da Costa, Paulo F. Flores, and José Monteiro. Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(6):1013–1026, 2008.

[5] Algirdas Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Trans. Electronic Computers*, 10(3):389–400, 1961.

[6] P. Cappello and K. Steiglitz. Some complexity issues in digital signal processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(5):1037–1041, Oct 1984.

[7] Oscar Gustafsson. Towards optimal multiple constant multiplication: a hypergraph approach. In *Conference Record of the Asilomar Conference on Signals Systems and Computers*, pages 1805–1809, Piscataway, NJ, 2008. IEEE.

[8] Martin Kumm. Optimal constant multiplication using integer linear programming. *IEEE Trans. on Circuits and Systems*, 65-II(5):567–571, 2018.

[9] Vincent Lefèvre. Multiplication by an integer constant vincent lefèvre n4192 mai 2001. 2001.

[10] Abdelkrim Kamel Oudjida, Nicolas Chaillet, and Mohamed Lamine Berrandjia. Radix-2^{r} arithmetic for multiplication by a constant: Further results and improvements. *IEEE Trans. on Circuits and Systems*, 62-II(4):372–376, 2015.

[11] Cadence Design Systems. Xcelium™Parallel Logic Simulator. `https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/xcelium-parallel-simulator.html`, 2017.

[12] Yevgen Voronenko and Markus Püschel. Multiplierless multiple constant multiplication. *ACM Trans. Algorithms*, 3(2):11, 2007.