Optimization of Gate-Level Area in High Throughput Multiple Constant Multiplications

Levent Aksoy
INESC-ID
Lisboa, Portugal

Eduardo Costa
Universidade Catolica de Pelotas
Pelotas, Brazil

Paulo Flores
INESC-ID/IST TU Lisbon
Lisboa, Portugal

José Monteiro
INESC-ID/IST TU Lisbon
Lisboa, Portugal

***Abstract*—This paper addresses the problem of optimizing gate-level area in a pipelined Multiple Constant Multiplications (MCM) operation and introduces a high-level synthesis algorithm, called HCUB-DC+ILP. In the HCUB-DC+ILP algorithm, initially, a solution with the fewest number of operations under a minimum delay constraint is found by the Hcub-DC algorithm. Then, the area around this local minimum point is explored exactly using a 0-1 Integer Linear Programming (ILP) technique that considers the gate-level implementation of the pipelined MCM operation. The experimental results at both high-level and gate-level clearly show the efficiency of HCUB-DC+ILP over previously proposed prominent MCM algorithms.**

I. Introduction

The MCM operation, that realizes the multiplication of a set of constants C by a single variable x, $y_j = c_j x$, is a central operation and performance bottleneck in many Digital Signal Processing (DSP) systems such as, Finite Impulse Response (FIR) filters and Discrete Cosine Transforms (DCT). Since the implementation of a multiplication operation in hardware is expensive in terms of area, delay, and power dissipation, the constant multiplications are generally realized using only addition, subtraction, and shift operations. Note that in bit-parallel MCM design, shifts can be realized using only wires without representing any area cost. Hence, an important and well-known optimization problem, called the MCM problem, is to realize the constant multiplications using the minimum number of addition/subtraction operations, which is NP-complete.

Over the years many efficient exact [1], [2] and approximate [3], [4] methods have been introduced for the MCM problem. The main idea behind these algorithms is to maximize the sharing of partial products that significantly reduces the number of operations and, consequently, the area of the MCM design. As a simple example, consider the constant multiplications $59x$ and $89x$ given in Figure 1(a). While the shift-adds realization of constant multiplications without partial product sharing leads to an MCM design with 5 operations (Figure 1(b)), the exact algorithm [2] finds a solution with 3 operations sharing the partial product $15x$ (Figure 1(c)).

In many DSP systems, delay and throughput are also crucial parameters, with circuit area being in many cases expandable in order to achieve these performance targets. The delay of the MCM design is generally considered as the number of adder-steps, which denotes the maximal number of adders/subtracters in series to produce any constant multiplication [5]. Thus,

This work was supported by the *Portuguese Foundation for Science and Technology* (FCT) research project Multicon - PTDC/EIA-EIA/103532/2008.

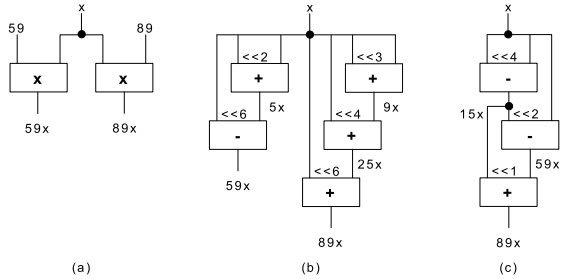

Fig. 1. (a) Constant multiplications $59x$ and $89x$; Their shift-adds realizations: (b) without partial product sharing; (c) with partial product sharing.

the MCM problem under a delay constraint is defined as finding the minimum number of operations that generate the constant multiplications without violating the delay constraint. Algorithms proposed for this problem can be found in [5], [6].

On the other hand, the throughput of a DSP system is generally increased by pipelining. The basic idea in pipelining is to overlap the processing of several tasks so that more intermediate tasks can be completed at the same time. For the pipelined realization of an MCM operation, an efficient design methodology was introduced in [7], [8]. In these methods, initially a set of addition/subtraction operations realizing the constant multiplications is found by a high-level algorithm. Then, the MCM operation is divided into n stages, where n is the number of adder-steps of the MCM design, and pipeline registers are inserted for each input of an operation at each stage. Also, the outputs of operations realizing the constant multiplications to be synthesized are carried to the last stage using pipeline registers. Figure 2(a) presents the pipelined realization of the MCM operation obtained by the exact algorithm [2] given in Figure 1(c) using this method.

Observe from Figure 2(a) that the gate-level area of a pipelined MCM design is related with the number of addition/subtraction operations. Hence, in [7], the authors improved the RAG-n algorithm [3] in order to obtain an MCM design with less number of operations. The gate-level area also depends on the number and size of the pipeline registers, that are related with the adder-step and bit-width of each operation realizing a constant multiplication. Hence, in [8], the authors proposed a high-level algorithm based on RAG-n, that focuses on synthesizing the constant multiplications at low logic depth by using partial products with low bit-widths.

However, the algorithms of [7], [8] do not take into account the implementation costs of adders, subtractors, and pipeline registers in hardware and they are not equipped with efficient partial product sharing heuristics [2], [4]. Hence, this paper ad-

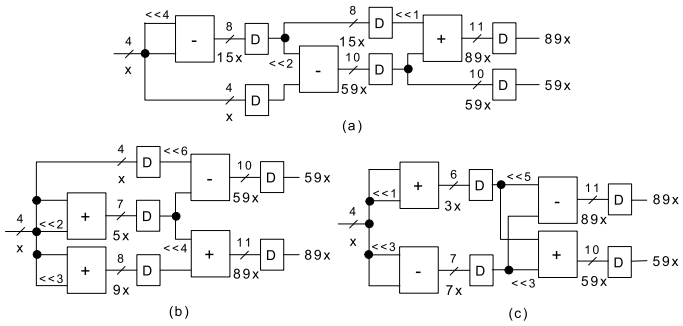

Fig. 2. Pipelined realizations of $59x$ and $89x$: (a) with the solution of [2]; (b) with the solution of Hcub-DC [6]; (c) with the solution of HCUB-DC+ILP.

dresses the gate-level area optimization problem in a pipelined MCM operation, which is defined as finding a set of operations that realizes the constant multiplications and yields a pipelined MCM design with optimal area at gate-level.

This paper proceeds as follows. Section II describes the HCUB-DC+ILP algorithm, experimental results are presented in Section III, and we conclude the paper in Section IV.

II. The Proposed Approach

The HCUB-DC+ILP algorithm consists of two efficient methods, the Hcub-DC [6] algorithm, *i.e.*, the modified version of Hcub [4] for the MCM problem under a delay constraint, and an *ILP* technique. In each iteration of HCUB-DC+ILP, while Hcub-DC finds a solution with the fewest number of operations respecting the minimum delay constraint on the MCM instance, the *ILP* technique formalizes the optimization of gate-level area problem in a pipelined MCM design as a 0-1 ILP problem on the solution of Hcub-DC. Returning to our example in Figure 2, the pipelined implementations of $59x$ and $89x$ obtained by the solutions of Hcub-DC and HCUB-DC+ILP are presented in Figures 2(b)-(c) respectively.

A. Implementation of the HCUB-DC+ILP Algorithm

In a preprocessing phase, the constants to be multiplied by a variable are converted to positive and then, made odd by successive divisions by 2. The resulting constants are stored without repetition in a set called target set T, the maximum bit-width of the target constants, bw, is determined, and the minimum adder-step of the MCM design, N, is computed as given in [5]. The HCUB-DC+ILP algorithm, whose pseudo-code is given in Figure 3, takes these parameters as an input.

In the iterative loop of HCUB-DC+ILP (lines 3-14), we initially find a set of operations, O, that implements the constant multiplications under the minimum adder-step constraint using Hcub-DC with a different seed at each time. Then, we determine the intermediate and target constants and their adder-step values in O and store them in a set called ready set, R. In order to increase the number of possible implementations of a constant in the *ILP* function, we add the depth-1 constants to R if they do not exist. The depth-1 constants have the adder-step value 1 and are in the form of $2^{i+1}-1$ and 2^i+1, where i ranges in between 1 and bw. To avoid unnecessary computations, we check if R is already included in R_{set} which is a set that stores all the ready sets given to the *ILP* function.

HCUB-DC+ILP(**T, bw, N**)

1: $seed = 0, icost_b = \infty$
2: $R_{set} \leftarrow \{\}$, $R_b \leftarrow \{\}$, $O_b \leftarrow \{\}$
3: **repeat**
4: $seed = seed + 1$
5: O = Hcub-DC(T, N, $seed$)
6: R = GenerateReadySet(O)
7: R = AddDepth1Constants(R, bw)
8: **if** $R \notin R_{set}$ **then**
9: $R_{set} \leftarrow R_{set} \cup R$
10: (R, O) = ILP(T, bw, R)
11: $icost$ = ComputeImplementationCost(O)
12: **if** $icost < icost_b$ **then**
13: $icost_b = icost$, $R_b \leftarrow R$, $O_b \leftarrow O$
14: **until** Termination conditions meet
15: **if** $N > 2$ **then**
16: R_b = AddDepth1Constants(R_b, bw)
17: R_b = AddDepth2Constants(R_b, bw, 50)
18: (R_b, O_b) = ILP(T, bw, R_b)
19: **return** O_b

Fig. 3. The HCUB-DC+ILP algorithm.

Then, a set of operations and its ready set are obtained by the *ILP* function. The MCM operation is pipelined as described in [8] and the cost of the pipelined MCM design, $icost$, is computed as if it were designed at gate-level with logic gates in the UMCLogic $0.18\mu m$ Generic II library. If its cost value is smaller than the best one ($icost_b$) found so far, then R_b, O_b, and $icost_b$ are updated. The iterative loop terminates whenever the number of elements in R_{set} reaches to 100[1] or the last 20[1] R obtained by Hcub-DC are identical. Finally, to explore further reductions in area of the pipelined MCM design, if N is greater than 2, the best ready set R_b is augmented with depth-1 constants and also with the smallest (in value) 50[1] depth-2 constants, which are the constants that can be realized using a single operation whose one of the inputs is a depth-1 constant and the other is 1 or a depth-1 constant. The *ILP* function is then applied to this ready set, and a set of operations that leads to a pipelined MCM design with optimal area is obtained. The main reason on considering the depth-2 constants only outside of the iterative loop is due to large 0-1 ILP problems, that increase significantly the runtime of the algorithm otherwise.

B. Implementation of the ILP Technique

The *ILP* technique used in HCUB-DC+ILP consists of four main parts: i) generation of constant implementations; ii) construction of the Boolean network that represents the implementations of constants; iii) formalization of the problem as a 0-1 ILP problem; iv) obtaining the minimum solution.

1) Generation of Constant Implementations: In the *ILP* function, initially, the elements of R are sorted according to their adder-step values in ascending order. Then, in an iterative loop, an element from R, r_i with its adder-step (stage) value d_{r_i}, is assigned to the output w of the *A-operation* [4], $w = |2^{ls_u}u + (-1)^s 2^{ls_v}v|2^{-rs_w}$, where: u and v are positive and odd input operands, $s \in \{0,1\}$ is the sign, which determines if an addition or a subtraction operation is to be performed; $ls_u, ls_v \geq 0$ are integers denoting left shifts of the operands; and $rs_w \geq 0$ is an integer indicating a right shift of the

[1]This value is determined empirically based on experiments.

result. Then, constants from R, r_j and r_k with $j,k < i$ and $max(d_{r_j}, d_{r_k}) + 1 \leq d_i$, are assigned to the inputs of the *A-operation*, u and v, and the values of s, ls_u, ls_v, and rs_w are searched exhaustively to realize r_i. Note that left shifts are allowed to be at most $bw+1$. Each *A-operation* implementing r_i is stored in a set called S_i and its implementation cost is computed as given in [9]. This procedure is repeated for each element of R. Note that each operation realizing a constant in the solution of Hcub-DC is also considered in this case, which ensures that the optimized pipelined MCM design will always have less or equal area as that obtained by Hcub-DC.

2) The Boolean Network: All possible implementations of elements of R are represented in a Boolean network that includes only AND and OR gates. While an AND gate represents an addition/subtraction operation implementing r_i, an OR gate combines all implementations of r_i in S_i. The outputs of the network are the outputs of OR gates representing the elements of T and its input is the input variable x denoted by 1.

Also, we need to include the optimization variables into the network so that the 0-1 ILP problem can be easily constructed. In proposed model, the optimization variables are associated with addition/subtraction operations and pipeline registers.

For each adder/subtractor denoted as an AND gate in the network, we generate an optimization variable, $opt_{a\pm b}$, where a and b stand for the inputs of the operation, and we include it to the inputs of the corresponding AND gate. The cost value of this type of optimization variable in the cost function is the gate-level area cost of the operation as computed in [9].

To determine the optimization variables associated with the pipeline registers, for each intermediate constant ic at the inputs of operations, we find the maximum of the adder-step values of operations, in which ic is one of the inputs, and obtain md_{ic}, as the minus 1 of this maximum value. Then, for each intermediate constant ic, we generate the optimization variables, $opt_{ic@d_{ic}}, opt_{ic@d_{ic}+1}, \ldots, opt_{ic@md_{ic}}$, with $d_{ic} > 0$. The cost value of this type of optimization variable in the cost function is determined as the implementation cost of a register at gate-level including $m + \lceil log_2 ic \rceil$ of D flip-flops, where m is the bit-width of the input variable x. Then, we generate a 2-input AND gate to include each of these optimization variables into the network. While the first input of this AND gate is the optimization variable denoting the constant ic at stage s, $opt_{ic@s}$, its second input is the output of the OR gate representing ic, if stage s is equal to d_{ic} or otherwise, the output of the AND gate that includes the optimization variable $opt_{ic@s-1}$. Note that s varies in between d_{ic} and md_{ic}. For the target constants, we do not need to generate such optimization variables, since their implementations are aimed.

Returning to our example in Figure 1, Figure 4 presents the network including the optimization variables constructed after a solution is obtained by Hcub-DC and depth-1 constants are added to R. Note that due to the space limitations, only a few implementations of target constants are shown in this figure. Also, 1-input OR gates for the intermediate constants 5, 7, and 9 are omitted and the type of an operation is shown inside of the AND gates representing adders/subtractors.

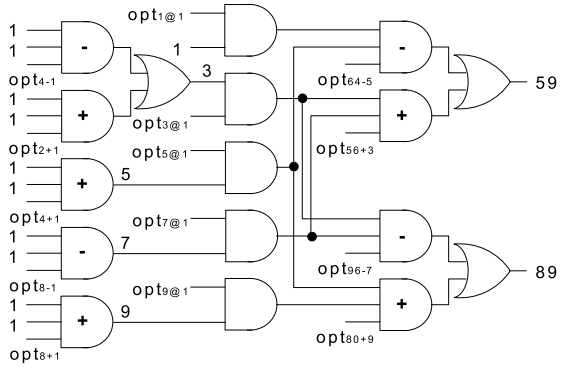

Fig. 4. The Boolean network generated for the target set $T = \{59, 89\}$.

3) The 0-1 ILP Formalization: The cost function of the 0-1 ILP problem is constructed as the linear function of optimization variables, where the cost value of each optimization variable is determined as described before. The constraints of the 0-1 ILP problem are obtained by finding the Conjunctive Normal Form (CNF) formulas of each gate in the network and expressing each clause of the CNF formulas as a linear inequality [10]. For example, a 2-input AND gate, $c = a \wedge b$, is translated to CNF as $(a+\overline{c})(b+\overline{c})(\overline{a}+\overline{b}+c)$ and converted to linear constraints as $a-c \geq 0$, $b-c \geq 0$, $-a-b+c \geq -1$. Also, the outputs of OR gates associated with the target constants are set to 1, since their implementations are aimed.

4) Finding the Minimum Solution: A generic 0-1 ILP solver will search the minimum value of the cost function on the generated 0-1 ILP problem. The set of operations that leads to the optimal area in a pipelined MCM design consists of the addition/subtraction operations whose optimization variables are set to 1 in the solution obtained by the 0-1 ILP solver.

III. Experimental Results

We compare HCUB-DC+ILP with prominent MCM algorithms [2], [6], [11] at both high-level and gate-level on multiplier blocks of FIR filters, whose coefficients are available at http://algos.inesc-id.pt/multicon/. Besides the algorithms of [2], [6], the MinLD algorithm [11] finds an MCM solution by synthesizing each constant at its minimum adder-step.

Table I presents the results of algorithms, where *oper*, *step*, and *icost* respectively stand for the number of operations, the number of adder-steps, and the implementation cost of a pipelined MCM design at high-level. Also, *CPU* denotes the runtime of Hcub-DC and HCUB-DC+ILP in seconds on a PC with Intel Xeon at 2.33GHz and 4GB memory.

Observe from Table I that although the DFSearch [2] finds a solution with the minimum number of operations, its solutions yield an MCM design with large number of adder-steps that consequently increases the area of a pipelined MCM design. As can be easily observed, low-complexity pipelined MCM designs are obtained when the delay constraint was set to the minimum adder-step of the MCM operation. Among these algorithms, HCUB-DC+ILP finds the best solutions in terms of *icost* indicating that the solutions of Hcub-DC can be improved for the pipelined MCM design. We note that the runtime of the 0-1 ILP solver SCIP 2.0 [12] in each iteration of HCUB-DC+ILP (line 10 of Figure 3) was at most 0.81s and was on

TABLE I
RESULTS OF HIGH-LEVEL SYNTHESIS ALGORITHMS.

Ins.	DFSearch [2]			MinLD [11]			Hcub-DC [6]				HCUB-DC+ILP			
	oper	step	icost	oper	step	icost	oper	step	icost	CPU	oper	step	icost	CPU
1	18	7	99722	22	3	68378	23	3	71746	0.1	22	3	64698	119.5
2	41	10	335356	52	3	156682	56	3	156812	0.3	52	3	143666	277.3
3	31	11	282414	37	3	123234	40	3	123212	0.1	39	3	112922	139.7
4	23	10	193192	28	3	88978	30	3	89162	0.2	29	3	81186	137.5
5	37	6	168256	44	3	137546	44	3	125870	0.1	43	3	117614	171.2
6	81	4	363824	83	3	276496	82	3	263652	0.1	82	3	255076	47.1
Tot.	231	48	1442764	266	18	851314	275	18	830454	0.9	267	18	775162	892.3

TABLE II
GATE-LEVEL RESULTS OF PIPELINED AND NON-PIPELINED MCM DESIGNS.

Ins.	Retiming			DFSearch [2]			MinLD [11]			Hcub-DC [6]			HCUB-DC+ILP			Non-pipelined		
	A	D	P	A	D	P	A	D	P	A	D	P	A	D	P	A	D	P
1	38.5	5.4	11	35.0	5.7	6	24.7	5.4	5	26.2	5.8	4	23.6	5.0	4	14.0	5.8	5
2	117.5	5.6	35	120.1	5.7	22	62.1	5.5	10	62.7	5.7	10	58.1	5.6	9	42.3	8.3	21
3	80.7	5.3	22	99.9	5.6	17	47.8	5.6	8	48.2	5.7	9	44.5	5.6	8	28.6	7.9	13
4	57.9	5.3	16	69.3	5.6	12	35.7	5.3	6	35.9	5.8	6	33.1	5.4	6	24.1	6.9	10
5	98.3	5.3	27	64.6	5.3	9	54.9	5.3	10	51.2	5.6	9	48.7	5.6	7	31.7	7.5	15
6	200.8	5.4	54	140.2	6.3	25	111.9	6.0	20	107.5	6.0	19	105.0	6.0	19	61.2	8.1	28
Tot.	593.6	32.3	165	529.1	34.2	91	337.1	33.0	58	331.6	34.6	59	313.1	33.2	53	201.9	44.5	93

average less than 0.5s. This is because of the small size of 0-1 ILP problems since the possible implementations of constants are limited to only the elements in the ready set. On the other hand, the runtime of the 0-1 ILP solver outside the iterative loop of HCUB-DC+ILP (line 18 of Figure 3) was maximum 63.7s and was 24.1s on average. This is because of larger 0-1 ILP problems due to the inclusion of depth-2 constants.

Table II presents the gate-level results of pipelined MCM operations obtained by the high-level synthesis algorithms given in Table I. It also introduces the gate-level results on pipelined realizations of constant multiplications using the logic synthesis tool enabling retiming. In this case, we defined the MCM operation as multiplications of constants by a variable and then, inserted n stage cascaded pipeline registers for each output of the MCM operation in VHDL description of the circuit, where n was 3, the same as in HCUB-DC+ILP. Moreover, it presents the gate-level results on non-pipelined (combinational) shift-adds implementations of the MCM designs based on the solutions of HCUB-DC+ILP. Note that the gate-level results were obtained using Cadence Encounter RTL Compiler with UMCLogic $0.18\mu m$ Generic II library. In Table II, A (mm^2), D (ns), and P (mW) indicate respectively the area, the delay in the critical path, and the RTL power estimation in MCM designs. The bit-width of the input variable was taken as 16 in this experiment.

Observe from Table II that the solutions of HCUB-DC+ILP also lead to pipelined MCM designs with minimum area at gate-level when compared to those synthesized using the solutions of prominent MCM algorithms. The maximum area improvement of HCUB-DC+ILP over Hcub-DC is obtained as 10.82% on the first instance. Also, although the direct pipelined realization of MCM operations enabling retiming increases the throughput slightly on average with respect to the results of high-level algorithms, the area and power dissipation values are significantly increased in this case. Moreover, when the pipelined and non-pipelined results of HCUB-DC+ILP are compared, it can be observed that, although pipelining increases the area of the design due to pipeline registers and the latency, 3 clock cycles on these instances, it not only increases the throughput of the MCM design but also decreases the power dissipation. This is because the glitching that occurs in reconvergent paths in a combinational MCM design is blocked by the pipeline registers placed at each stage of the pipelined MCM design.

IV. CONCLUSIONS

This paper presented a high-level algorithm for the optimization of gate-level area in pipelined MCM designs. The proposed approach can incorporate any MCM heuristic and can be applied to various optimization problems, such as the optimization of gate-level area in bit-parallel and digit-serial MCM designs, by only changing the 0-1 ILP formalization.

REFERENCES

[1] O. Gustafsson and L. Wanhammar, "ILP Modelling of the Common Subexpression Sharing Problem," in *ICECS*, 2002, pp. 1171–1174.

[2] L. Aksoy, E. Gunes, and P. Flores, "Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate," *Elsevier Journal on Embedded Hardware Design*, vol. 34, pp. 151–162, 2010.

[3] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *TCAS II*, vol. 42, no. 9, pp. 569–577, 1995.

[4] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.

[5] H.-J. Kang and I.-C. Park, "FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders," *IEEE TCAS II*, vol. 48, no. 8, pp. 770–777, 2001.

[6] Spiral website, http://spiral.ece.cmu.edu/mcm/gen.html.

[7] U. M.-Baese, J. Chen, C.-H. Chang, and A. Dempster, "A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters," in *APCCS*, 2006, pp. 1555–1558.

[8] K. Macpherson and R. Stewart, "RAPID PROTOTYPING - Area Efficient FIR filters for High Speed FPGA Implementation," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 153, no. 6, pp. 711–720, 2006.

[9] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Optimization of Area and Delay at Gate-Level in Multiple Constant Multiplications," in *EUROMICRO DSD*, 2010, pp. 3–10.

[10] P. Barth, "A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization," MPII, Tech. Rep., 1995.

[11] M. Faust and C.-H. Chang, "Minimal Logic Depth Adder Tree Optimization for Multiple Constant Multiplication," in *ISCAS*, 2010, pp. 457–460.

[12] Solving Constraint Integer Programs website, http://scip.zib.de/.