Hybrid Multiple Constant Multiplication for FPGAs

Martin Kumm and Peter Zipf
Digital Technology Group
University of Kassel, Germany
Email: {kumm, zipf}@uni-kassel.de

***Abstract*—The multiple constant multiplication (MCM) is a frequently used operation in many digital signal processing applications like digital filters. Mapping MCM to modern heterogeneous field programmable gate arrays (FPGAs) is commonly done by either using embedded multipliers or a carry-chain dominated method using additions, subtractions and bit shifts. The use of embedded multipliers is restricted by their quantity and word size. In particular, large coefficients as required for floating point MCM may need lots of embedded multipliers for each single constant. This work presents an optimization method which is able to include a user-defined number of embedded multipliers into a fully pipelined add/shift based MCM operation. Multiplier results can be shared between several constants for resource reduction. The algorithm can be either used for finding a trade-off between DSP and logic resources or to realize large MCM blocks with less DSP resources.**

I. Introduction

Multiple constant multiplication (MCM), where a variable is multiplied by several constants, is a commonly used operation in digital signal processing (DSP). It is used, e. g., in digital filters [1]–[5], constant matrix multiplications [6] and discrete transforms like the fast Fourier transform (FFT) [7]. Lots of research has been done so far to reduce the complexity of MCM circuits. The MCM operation is usually reduced to additions, subtractions and bit shifts which are arranged in a so called adder graph. Finding a configuration with minimal number of adders and subtractors (shifts are assumed to be free) is known as MCM problem [8]. Although primary targeted to application specific integrated circuits (ASICs) it was shown that the same principles are the best choice to reduce logic for FPGAs [1]–[5] as they outperforms previous FPGA methods like distributed arithmetic [1].

However, using MCM blocks on FPGAs with an appropriate speed requires that the adders are pipelined [3]. Algorithms for directly optimizing pipelined adder graphs were proposed recently [4], [9]. A different MCM method using the LUT resources of FPGAs was used in [10], [11]. It is beneficial for small input word length in the order of 8 bit but is likely to consume more resources than MCM for larger word length.

In contrast to these adder graph and LUT-based approaches, the multiplication intensive digital filters were one of the main driving forces for embedding multipliers and DSP blocks into modern heterogeneous FPGAs. So, one has to ask why multiplications should be reduced to additions, subtractions and shifts when modern FPGAs support more and more embedded multipliers for that. There are two main limitations when using embedded multipliers: First, they are a limited resource. Second, the multiplier word size is limited which may drastically raises the number of required embedded multipliers for larger word sizes.

Different methods were proposed for computing large multiplications which typically appear in floating-point MCM operations [12] using small multipliers [13], [14]. Mantissa multipliers of 24 bit and 53 bit are necessary for single and double precision, respectively. Both cases do not fit even in state-of-the-art FPGA DSP blocks which are 17×24 bit for Xilinx Virtex 5/6/7 and 18×18 bit for Altera Stratix III/IV (both unsigned), so an appropriate tiling [13], [14] has to be found. A mantissa multiplier for single precision floating point can be constructed by adding the result of two 17×24 DSP blocks where one of the inputs is split into two sub words:

$$a \cdot b = a \cdot b^L + 2^{17} a \cdot b^H \tag{1}$$

A tiling for a double precision 53×53 bit mantissa multiplier was recently proposed in [14] where the number of 17×24 DSP blocks could be reduced from 10 (Xilinx Coregen) to 7. However, in the MCM case with constant multipliers there is much room for improvement as intermediate results from embedded multipliers can be shared to reduce resources.

Therefore, an algorithm to optimize hybrid pipelined adder graphs (PAGs) is proposed which is able to include a user-defined number of embedded multipliers. It is based on the recently proposed reduced pipelined adder graph (RPAG) algorithm [9] which optimizes ordinary PAGs. It was already used for the special case of optimizing floating point MCM blocks [12] in which single precision 24 bit mantissa multipliers were reduced to 17 bit DSP blocks. The proposed algorithm is a generalization of both methods and can be used to treat the two limitations mentioned above to either efficiently implement missing embedded multipliers by logic and/or to reduce the number of embedded multipliers for large word size MCM operations. In the next section, the underlying RPAG algorithm [9] is briefly introduced which is then extended to the hybrid algorithm in Section III.

II. Optimizing Multiplier blocks using RPAG

The RPAG algorithm [9] optimizes MCM blocks using PAGs. A simple example PAG is shown in Fig. 1 which realizes the multiplication with constants from the set $\{39, 114, 168\}$. Each node in a PAG corresponds to a registered adder, subtractor or a pure register. This is formally represented as $\mathcal{A}$-operation [8], which is defined as

$$\mathcal{A}_q(u, v) = |2^{l_1} u + (-1)^{sg} 2^{l_2} v| 2^{-r} \tag{2}$$

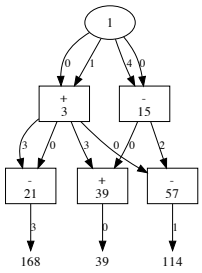

Fig. 1. Example Pipelined Adder Graphs of coefficient set $\{39, 114, 168\}$

with $q = (l_1, l_2, r, sg)$, where u and v are the input arguments, l_1, l_2 and r are shift factors and the sign bit $sg \in \{0, 1\}$ denotes whether an addition or subtraction is performed. All edge weights correspond to shift factors l_1 and l_2, e. g., node 15 is realized by left shifting the input x by 4 bits and subtracting the unshifted input: $2^4x - 2^0x = 15x$.

The most complex task in optimizing PAGs is finding valid intermediate node values, the shift values can be obtained easily in a second step by evaluating (2). Therefore, a set X_s of node values is defined for each pipeline stage s to describe a solution of a PAG. The pipeline sets for the PAG in Fig. 1 are, e. g., $X_0 = \{1\}$, $X_1 = \{3, 15\}$ and $X_2 = \{39, 114, 168\}$. The number of pipeline stages is set to $S := \max_{t \in T} \text{AD}_{\text{min}}(t)$, where $\text{AD}_{\text{min}}(t)$ is the minimal adder depth of t which is defined as the number of adder stages needed to compute a coefficient t out of the target set T. The minimal adder depth can be computed using

$$\text{AD}_{\text{min}}(x) = \lceil \log_2(\text{nz}(x)) \rceil \tag{3}$$

where $\text{nz}(x)$ is the number of nonzero digits of x in minimum signed digit (MSD) representation [9].

An outline of RPAG is given in the following as basis for the extension to the hybrid algorithm described in Section III, a more detailed description can be found in [9]. In contrast to most existing MCM algorithms, RPAG searches from the output nodes to the input node 1 in a greedy manner:

1) The algorithm starts from stage $s = S$ and X_S is initialized to the odd representations of T.
2) Valid elements of $p \in X_{s-1}$, called predecessors, are evaluated. A valid predecessor must fulfill

$$\text{AD}_{\text{min}}(p) < s \tag{4}$$

to reduce the adder depth in each stage until it is zero (for node 1). The evaluation is started with single predecessors which are realized by one of the three possible topologies of Fig. 2 (a)-(c):
   a) Element $w \in X_s$ has a lower adder depth than s and can be copied from p using a register
   b) w can be computed by $p \cdot (2^k \pm 1)$ with $k \in \mathbb{N}$
   c) w can be obtained by an $\mathcal{A}$-Operation using p and another element of X_{s-1} which was already realized

   If any predecessor was found, the best one from which the maximum number of elements in X_s can be computed with minimal costs [9] is copied to X_{s-1} and the algorithm continues with step 4.
3) No single predecessor was found and the algorithm evaluates the pairs (p_1, p_2) of predecessors, shown in Fig. 2 (d), obtained from the MSD representation of w [9] and copies the best pair to X_{s-1}.
4) If there are no elements in X_s left that can be computed from X_{s-1}, reduce s by one. If $s \neq 1$ continue with step 2, if $s = 1$ all sets $X_{1...S}$ are computed (X_0 is known to be $\{1\}$).

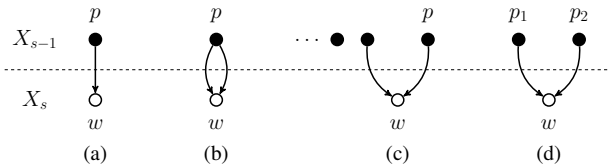

Fig. 2. Predecessor graph topologies for (a)-(c) a single predecessor p, (d) a predecessor pair (p_1, p_2)

III. The Hybrid Adder/Multiplier Optimization

A. Incorporating Multipliers in MCM Blocks

The target of our work is to incorporate a user-defined number of embedded multipliers in the PAG, such that the costs of the total hybrid PAG are minimal. An example hybrid PAG is shown in Fig. 3. It can be separated into three blocks: 1) the input PAG, 2) the embedded multipliers, both with S_i pipeline stages, and 3) the output PAG with S_o stages. Hence, the total hybrid PAG consists of $S = S_i + S_o$ pipeline stages. The input PAG is used to compensate for (possibly) insufficient embedded multipliers while the output PAG is used to compute output coefficients with large word size from previous adder and multiplier results.

For that purpose, an additional word size (w_s) constraint is included in the RPAG algorithm for predecessor selection. Both word size and adder depth (4) constraints can be set to arbitrary values depending on the stage of the hybrid PAG. Constraining the word size always constrains the maximum AD. Thus, the word size constraint is more rigorous in limiting the search space than the AD constraint.

The hybrid PAG can now be divided into four segments as shown on the right side of Fig. 3 with different constraints listed in Table I. The first segment corresponds to the first stages of the input PAG where only the adder depth is constrained to the actual stage s. In segments II-IV, only the word size is constrained. Clearly, in segment II (stage S_i) it must be constrained to the word size of the embedded multipliers w_m. In each of the following stages, the word size can be maximally doubled, leading to the upper bound $w_s^{\max}$ in Table I. From the output side (segment IV), it is obvious that the word size constraint must be equal to the target word size w_T. As the target word size can be maximally halved in each of the previous stages, this leads to the lower bound $w_s^{\min}$. As long as w_T is not dividable by w_m we have a degree of freedom in choosing w_s. Lower w_s reduces the number of

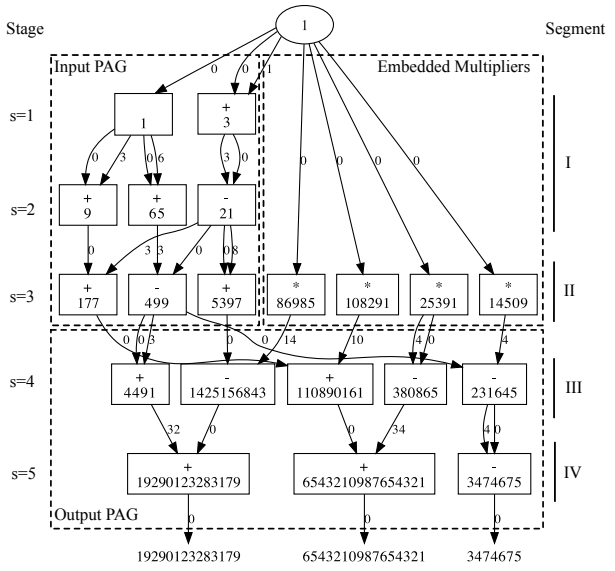

Fig. 3. A pipelined hybrid adder graph consisting of input adder graph, embedded multipliers ($S_i = 3$) and output adder graph ($S_o = 2$)

elements at lower stages, and vice versa. Therefore, lower w_s tend to a lower number of embedded multipliers at the cost of additional adders.

When RPAG reaches stage S_i, elements that are realized by embedded multipliers are selected and removed from X_{S_i} before proceeding with step 2 of the algorithm. The selection is done by choosing the elements of X_{S_i} with the highest adder depth as these are the most costly elements to realize with the input PAG. If possible, the depth of the input PAG is reduced after the reduction of X_{S_i}. In the rest of this section, we will derive the minimal necessary number of stages for the input and output PAGs.

B. Depth of the Input PAG

According to (3), the depth of the input PAG depends on the number of non-zero digits of the constant. The maximal number of non-zeros of a constant c with word size w can be obtained by setting each second bit to a non-zero value, i. e., setting $c = 1010\ldots010$ leads to $\text{nz}_{\max}(x) = w/2$. Hence, the maximum adder depth, needed to replace a w bit multiplier is:

$$\text{AD}_{\max}(w) = \lceil \log_2(w) \rceil - 1 \tag{5}$$

The same results from an odd number of bits in c. Therefore, every multiplier with word size w_m can be replaced by a PAG with at most $S_i = \text{AD}_{\max}(w_m)$ stages. Note that the reverse statement can not be given as a single adder stage can realize arbitrary large word sizes.

C. Depth of the Output PAG

The objective of the output PAG is to compute the target coefficients with word size $w_T > w_m$ (otherwise, the output PAG is not necessary). The input word size of the output PAG is w_m and the maximum output coefficient word size w_T from target set T is given by $w_T = \max_{t \in T} \lceil \log_2(t) \rceil$. The word size of an arbitrary constant can be doubled using one stage of adders. Hence, the number of stages needed for the output PAG is given by $S_o = \lceil \log_2(w_T / w_m) \rceil$.

TABLE I
CONSTRAINTS FOR ADDER DEPTH AD_s AND WORD SIZE (w_s^{MIN} MIN. AND w_s^{MAX} MAX. POSSIBLE WORD SIZE) OF SEGMENTS SHOWN IN FIG. 3

Segment	s	AD_s	$w_s^{\min}$	$w_s^{\max}$
I	$0 \ldots S_i - 1$	s	0	∞
II	S_i	∞	w_m	w_m
III	$S_i + 1 \ldots S_o - 1$	∞	$\left\lceil \frac{w_T}{2^{S-s}} \right\rceil$	$2^{s-S_i} w_m$
IV	$S = S_i + S_o$	∞	w_T	w_T

TABLE II
SYNTHESIS RESULTS OF BENCHMARK SET [2]

		$M =$ 0		$\lfloor \frac{1}{4} N_{\text{uq}} \rfloor$		$\lfloor \frac{1}{2} N_{\text{uq}} \rfloor$		$\lfloor \frac{3}{4} N_{\text{uq}} \rfloor$		N_{uq}	
N	N_{uq}	Add/ Reg	Slices	Add/ Reg	Slices	Add/ Reg	Slices	Add/ Reg	Slices	Add/ Reg	Slices
6	2	10	42	10	42	7	27	7	27	3	13
10	4	13	60	10	47	7	25	5	20	3	10
13	6	18	84	14	65	9	41	6	26	3	13
20	8	19	95	13	60	10	46	5	19	3	11
28	13	23	114	20	96	16	80	9	37	3	16
41	20	32	166	25	125	16	84	9	47	3	10
61	30	53	218	34	174	24	120	15	73	3	14
119	53	70	334	52	241	34	153	16	61	3	13
151	70	86	401	69	320	43	204	24	113	3	14
avg.:		36.0	168.2	27.4	130.0	18.4	86.7	10.7	47.0	3	12.7

IV. RESULTS

In this Section, two experiments are presented to evaluate the performance of the proposed method. For that, VHDL code generators for the different architectures were written and synthesis results were obtained using Xilinx ISE 13.4 for the Virtex-6 FPGA XC6VSX475T-1 (after place & route). All MCM blocks were designed with full-precision. All dynamic power analyses were obtained for a uniformly distributed random input sequence using XPower.

A. Slice Ressources vs. Embedded Multipliers

In the first experiment, low word size benchmark coefficients from [2]–[4] were optimized for different numbers of embedded multipliers M. The results are shown in Table II. Here, N is the number of filter taps and N_{uq} the number of the unique, positive, odd coefficients excluding 1. M was equally varied between 0 and N_{uq} in 5 steps. A detailed pareto front including the dynamic power dissipation for filter instance $N = 28$ is shown in Fig. 4. It can be observed, that a steady and nearly linear reduction of slice resources can be achieved by using more embedded multipliers. However, this is paid with an increased dynamic power of up to 61%.

B. Reducing Embedded Multipliers in Large MCM Blocks

In the second experiment, hybrid PAGs were used to reduce embedded multipliers in single and double floating-point mantissa MCM blocks. Coefficient sets of seven FIR filters of different length served as a realistic benchmark set. They were designed using the Parks-McClellan algorithm. All filters

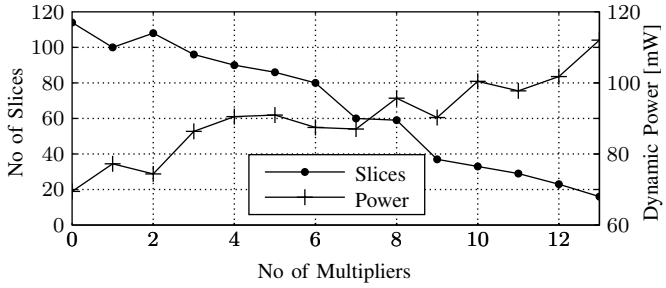

Fig. 4. Pareto front for filter $N = 28$ of benchmark set [2]

are low pass filters with a normalized passband and stopband frequency of $f_p = 0.3$ and $f_s = 0.4$, respectively. The number of taps varies from 10 to 120 resulting in passband/stopband approximation errors from $-15\,\text{dB}$ to $-100\,\text{dB}$. Due to the symmetry of the coefficients, the number of unique coefficients N_{uq} is half the number of taps. Synthesis and dynamic power results for single and double precision MCM blocks are listed in Table III and Table IV, respectively. For comparison, reference designs using state-of-the-art methods were implemented.

For single precision, an MCM block where each constant is realized using two 17×24 DSP blocks as given in (1) was implemented as reference. In general, the same number of adders are used but for large N_{uq}, only half of the DSP blocks are necessary. However, the adders of the conventional method can be completely mapped into one DSP48E(1) slice of Virtex-5/6/7, which contains a 48 bit output adder including a 17 bit input shift. These ouput adders can only be used when the result of a DSP block is not used by other coefficients as each DSP slice has only one output. Hence, halving the DSP count is paid by roughly doubling the slices. The dynamic power of the proposed method is reduced by 8.4%.

For double precision, an MCM block which uses the tiling of Banescu et al. [14] for each coefficient was used for reference, which can be generated by the FloPoCo core generator[a]. Interestingly, one of the seven multipliers of that tiling was realized using slice logic. RPAG was configured to reduce the constants to 24 bit, such that a 53×24 bit multiplier consisting of 3 DSP blocks and some logic can be used. As can be seen from Table IV, slice, DSP and power reductions up to 39%, 24% and 35% could be achieved, respectively. The higher the number of coefficients the more reductions can be achieved.

V. Conclusion

A flexible algorithm for the hybrid optimization of multiple constant multiplication operations using embedded multiplier and logic resources of modern heterogeneous FPGAs was presented. It was shown that significant resource and power reductions exceeding the best results reported in literature can be achieved. Due to a user-defined multiplier constraint, MCM blocks can be efficiently mapped to any FPGA size as insufficient embedded multipliers are realized by carry-chains. Beyond that, the method can be used to reduce the amount of embedded multipliers in large MCM blocks which typically appear in floating-point MCM operations.

[a]Available at http://flopoco.gforge.inria.fr

TABLE III
MCM benchmark with single precision mantissas

	MCM using conventional tiling					Hybrid RPAG (proposed)				
N_{uq}	Add.	DSP	Slices	Pow. [mW]	f_{max} [MHz]	Add.	DSP	Slices	Pow. [mW]	f_{max} [MHz]
5	5	10	25	53	252	5	7	62	55	332
10	10	20	50	89	239	10	17	117	97	308
20	20	40	100	154	240	20	23	238	150	297
30	30	60	150	221	224	30	30	353	204	281
40	40	80	200	293	220	40	40	451	260	277
50	50	100	253	363	212	50	50	578	323	252
60	60	120	313	436	212	60	59	696	384	334
avg.:	30.7	61.4	155.9	229.8	229.0	30.7	32.3	356.4	210.4	297.7

TABLE IV
MCM benchmark with double precision mantissas

	MCM using tiling of [14]					Hybrid RPAG (proposed)				
N_{uq}	Add.	DSP	Slices	Pow. [mW]	f_{max} [MHz]	Add.	DSP	Slices	Pow. [mW]	f_{max} [MHz]
5	25	30	951	430	203	15	30	642	286	221
10	50	60	1704	789	196	30	60	1233	538	227
20	100	120	3411	1659	192	60	111	2441	1013	195
30	150	180	5784	2539	178	90	171	3707	1616	173
40	200	240	7483	3471	175	120	204	4489	2251	172
50	250	300	9411	4181	177	150	243	5684	2948	135
60	300	360	10403	5200	149	180	273	6367	3148	169
avg.:	153.6	184.3	5592.4	2610	181.8	92.1	156.0	3509.0	1686	185.3

References

[1] U. Meyer-Baese, J. Chen, C.-H. Chang, and A. G. Dempster, "A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters," in *IEEE APCCAS 2006*, 2006, pp. 1555 – 1558.

[2] S. Mirzaei, R. Kastner, and A. Hosangadi, "Layout aware optimization of high speed fixed coefficient FIR filters for FPGAs," *Int. Journal of Reconfigurable Computing*, vol. 3, pp. 1–17, Jan 2010.

[3] M. Kumm and P. Zipf, "High speed low complexity FPGA-based FIR filters using pipelined adder graphs," in *Field Programmable Technology, Int. Conf. on (ICFPT)*, 2011.

[4] U. Meyer-Baese, G. Botella, D. Romero, and M. Kumm, "Optimization of high speed pipelining in FPGA-based FIR filter design using genetic algorithm," in *SPIE Defense Security+Sensing*, 2012.

[5] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Design of low-complexity digital finite impulse response filters on FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, 2012, pp. 1197–1202.

[6] L. Aksoy, E. Costa, and P. Flores, "Multiplierless Design of Linear DSP Transforms," *IFIP Advances in Information and Communication Series*, vol. 379, pp. 73–93, Jan. 2012.

[7] F. Qureshi and O. Gustafsson, "Low-complexity reconfigurable complex constant multiplication for FFTs," in *Circuits and Systems, IEEE Int. Sym. on (ISCAS)*, 2009.

[8] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Alg.*, vol. 3, no. 2, 2007.

[9] M. Kumm, M. Faust, P. Zipf, and C.-H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Circuits and Systems, IEEE Int. Sym. on (ISCAS)*, 2012.

[10] M. Wirthlin, "Constant coefficient multiplication using look-up tables," *The Journal of VLSI Signal Processing*, 2004.

[11] M. Faust and C.-H. Chang, "Bit-parallel Multiple Constant Multiplication using Look-Up Tables on FPGA," *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pp. 657–660, 2011.

[12] M. Kumm, K. Liebisch, and P. Zipf, "Reduced complexity single and multiple constant multiplication in floating point precision," in *Field Program. Logic and App., Int. Conf. on (FPL)*, 2012.

[13] F. de Dinechin and B. Pasca, "Large multipliers with fewer DSP blocks," in *Field Program. Logic and App., Int. Conf. on (FPL)*, 2009.

[14] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for floating-point double precision and beyond on FPGAs," *ACM SIGARCH Comp. Arch. News*, vol. 38, 2011.