MULTIPLE CONSTANT MULTIPLICATION WITH TERNARY ADDERS

Martin Kumm[†]*, Martin Hardieck*[†]*, Jens Willkomm*[†]*, Peter Zipf*[†]*, Uwe Meyer-Baese*[‡]

[†]University of Kassel, Digital Technology Group, Germany
[‡]Florida State University, Department of Electrical & Computer Engineering, Tallahassee, FL USA
Email: kumm@uni-kassel.de, martin.hardieck@student.uni-kassel.de,
jens.willkomm@student.uni-kassel.de, zipf@uni-kassel.de, umeyerbaese@fsu.edu

ABSTRACT

The scaling operation, i. e., the multiplication with a single constant is a frequently used operation in many kinds of numeric algorithms. The multiple constant multiplication (MCM) is a generalization where a variable is multiplied by several constants. This kind of operation is heavily used, e. g., in digital filters or discrete transforms. It was shown in recent work that small, fast and power efficient MCM implementations can be realized by using the fast carry chains of FPGAs rather than wasting specialized embedded multipliers. However, in the work so far, only common two-input adders were used. As FPGAs today support ternary adders, i. e., adders with three inputs, this work investigates the optimization of pipelined MCM circuits which include ternary adders. It is shown experimentally that 27% less operations are needed on average by using ternary adders, resulting in 15% slice (Xilinx) and 10% ALM (Altera) reductions, respectively.

1. INTRODUCTION

The multiplication of a variable with several constants, the multiple constant multiplication (MCM), is an important operation in digital signal processing. It is mostly motivated by digital filters [1–8], but can be also found in applications which include constant matrix multiplications [9], e. g., discrete transforms like the fast Fourier transform (FFT) [10]. Of course, MCM is a special case of single constant multiplication (SCM) which occurs in nearly every numeric algorithm. Many different methodologies for realizing MCM operations or digital filters on FPGAs have been proposed over the years:

a) Directly using embedded multipliers
b) Distributed arithmetic (DA) [1, 7, 8]
c) Reduction of the constant multiplication to additions, subtractions and bit shifts [1–5, 11]
d) Look-up table (LUT)-based multiplication [12–14]

Before embedded multipliers became available, distributed arithmetic (method b) was the first choice for implementing digital filters on FPGAs as the convolution operation is realized by several look-up tables (LUTs), which can be directly mapped to the FPGA LUTs.

Embedded multipliers (method a) was the answer of the FPGA manufacturers for the high demand of multipliers used in digital signal processing. According to the corresponding manuals [15, 16], digital filters were one of the main driving forces to push embedded multipliers or DSP blocks into the FPGA fabric. However, as usual for those specialized coarse-grained blocks, they are limited in quantity and are inflexible in word size.

In method c), the constant multiplications are reduced to additions, subtractions and bit shifts which can be represented with so called adder graphs. Finding an adder graph with a minimal number of nodes (i. e., adders and subtractors) is known as MCM problem [2]. It was shown that the adder graph method leads to considerably smaller circuits compared to DA. About 50% resource reductions were reported for Altera [1] and Xilinx [3] devices. However, to achieve the same speed as DA, these adder graphs have to be pipelined [1, 3]. As the addition of pipeline registers may significantly increase the complexity, a resource-optimal scheduling of pipeline registers for a given adder graph was proposed [4]. Nevertheless, directly optimizing the adder graph for pipelining is highly advantageous. Algorithms for optimizing pipelined adder graphs (PAGs) were proposed recently [5, 11]. Of course, PAGs can be combined with embedded multipliers to replace insufficient multiplier resources (e. g., on low cost FPGAs) or to realize larger word sizes [17]. As a side effect, the dynamic power consumption of PAGs is reduced compared to the embedded multipliers (DSP48E) on Xilinx Virtex 6 FPGAs for which XPower reported up to 61% more power than for the equivalent PAG circuit [17].

An interesting alternative for constant multiplication which also uses the FPGA-LUTs was proposed by Chapman [12], further extended by Wirthlin [13] and adapted to MCM by Faust et al. [14]. They could show that for a small

input word size of 8 bit, its resource consumption is comparable to (unpipelined) adder graph realizations but have shorter delays.

To summarize the discussed methods above, the PAG method is the one which usually leads to the least slice resources, low power and high performance without requiring embedded multipliers. As the latest FPGA families of Altera and Xilinx support ternary adders, i. e., adders with three inputs, this paper investigates the optimization of PAGs which include ternary adders. To implement ternary adders, the LUTs are used in addition to the FPGA carry logic. As the LUTs of the two-input adders/subtractors in a PAG are not used for any additional logic function, the ternary adders come with no extra cost and can be used to realize PAGs with less complexity. For this, a novel optimization heuristic is proposed in this work which is based on the reduced pipelined adder graph (RPAG) algorithm [11]. Both implementations are available online [18].

In the following, the realization of ternary adders on different FPGA families is discussed, followed by a resource and speed comparison on Altera and Xilinx FPGAs. In Section 3, the reduced pipelined adder graph algorithm with ternary adders (RPAGT) is described in detail followed by experimental results in Section 4. A conclusion is drawn in Section 5.

2. REALIZATION OF TERNARY ADDERS

A ternary adder has to perform a summation of the form $s = x + y + z$. This can simply be realized by two stages of ripple carry adders (RCAs) where the first stage computes $s' = x + y$ and the second stage computes $s = s' + z$. Alternatively, the idea of a carry save adder (CSA) tree can be used to compute the same result. Then, the full adders (FAs) of the first stage can be used to realize a 3:2 compressor, i. e., the three input bit vectors are compressed to two bit vectors which are obtained by the sum and carry outputs. These two bit vectors can then be merged using a common RCA, which is sometimes called vector merging adder (VMA). At first view, this saves no hardware resources as the same number of full adders are used, but on modern FPGAs, the 3:2 compressors can be mapped to otherwise unused resources of the FPGA LUT.

2.1. Realization on Altera FPGAs

The adaptive logic modules (ALMs), available on Alteras Arria I,II,V and Stratix II-V FPGAs directly support ternary adders. For this, the ALM has to be configured to the shared arithmetic mode [19] in which the output of one LUT is connected to the FA input of the next higher bit. Two four-input LUTs are available for each adaptive LUT (ALUT) such that the 3:2 compressor can be directly mapped to these LUTs, realizing sum $s'_i = x_i \oplus y_i \oplus z_i$ and carry $c'_i = x_i y_i + x_i z_i + y_i z_i$. The resulting ternary adder structure is shown in Fig. 1(a). With each ALM, two output bits can be computed. Note that a two-input adder with the same output word size requires the same number of ALMs. The Quartus II tool automatically detects ternary adders from a high-level hardware description language (HDL), e. g., a VHDL statement of the form `s <= x+y+z`. Unfortunately, the ternary subtract operations ($x-y+z$, $x+y-z$ and $x-y-z$) are not directly supported from a high-level description. To do so, the corresponding input(s) must be inverted and the carry-in signal(s) must be set to 1. To set the carry bit(s), the word size of the adder has to be extended by one bit (for $x-y+z$ and $x+y-z$) or two bit (for $x-y-z$), resulting in one or two additional ALUTs.

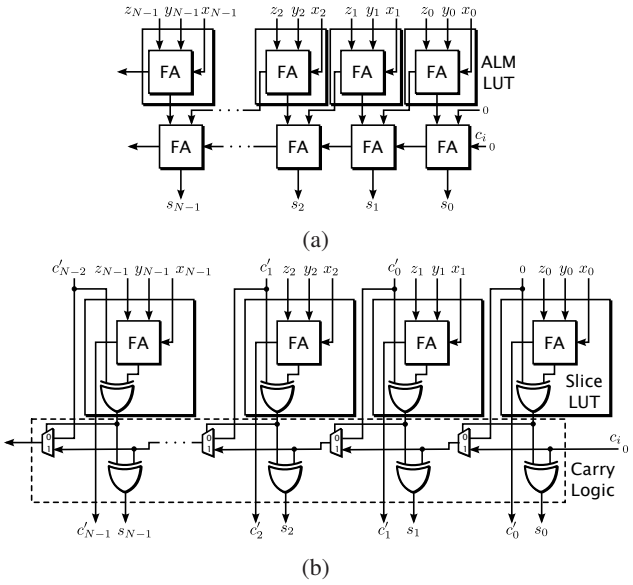

Fig. 1: Realization of ternary adders on (a) Altera Stratix II-V ALMs (b) Xilinx Virtex 5-7 Slices

2.2. Realization on Xilinx FPGAs

On Xilinx FPGAs, the situation is a bit more complicated as the slice logic does neither provide real full adders nor the necessary internal slice connection from one LUT to the full adder input of the next higher bit. However, a ternary adder is possible on modern FPGA families which provide six-input LUTs [20], namely the Virtex 5-7, Spartan 6, Kintex 7 and Artix 7 families. To extend the fast carry logic to a real full adder, one XOR gate has to be realized in the LUT. In addition to that, the full adder for the 3:2 compressor is realized in the same LUT. The carry output of that full adder can now be routed to the next higher FA using the FPGA routing fabric. The resulting slice configuration is shown in Fig. 1(b). The multiplexer and the XOR gate shown in the dashed box belong to the carry logic of the slice. The FA and XOR gate shown in the solid boxes are realized in the

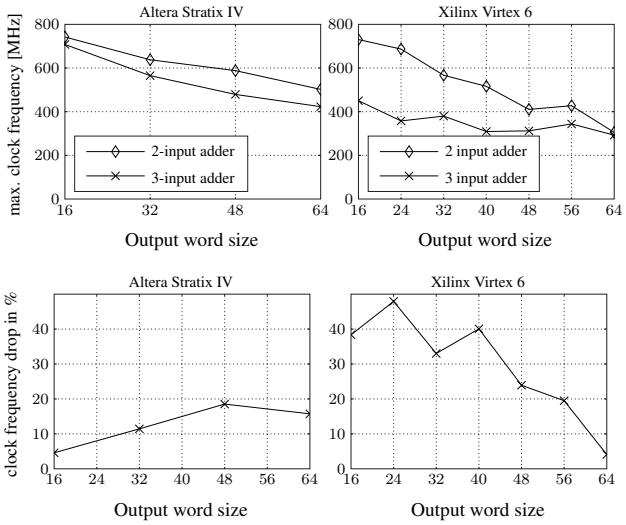

Fig. 2: Comparison of clock frequencies of two-input and ternary adders (top) and the frequency drop from two to three inputs (bottom) realized on Virtex 6 and Stratix IV

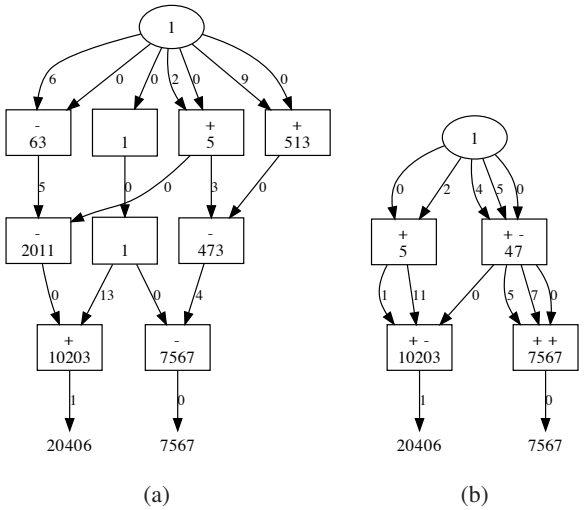

Fig. 3: Pipelined adder graphs of a constant multiplier with coefficients $\{7567,20406\}$ using (a) two-input adders (b) ternary adders

slice LUTs, where output O6 is used for the sum and O5 is used for the carry (c_i') of the 3:2 compressor. With this architecture, four output bits can be computed in each slice. Like for Altera, the same number of slices are required for a two-input adder with the same output word size. Currently, there is no guarantee that ternary adders from a high-level HDL description are mapped in that efficient way by the Xilinx tools. Thus, a ternary adder was built by using the primitives (`CARRY4` and `LUT6_2`). Note that all ternary subtract operations ($x-y+z$, $x+y-z$ and $x-y-z$) can be realized without additional resources. Both implementations (Altera and Xilinx) were published at OpenCores [21].

2.3. Adder Comparison

As the ternary adders have more combinatorial logic in their critical path it is expected that their speed is reduced. Hence, the maximum clock frequencies for adders with two and three synchronized inputs and synchronized outputs were obtained by synthesis experiments. This was done for Altera Stratix IV (EP4SGX230KF40C2) using Quartus-II 10.1 and Xilinx Virtex 6 (XC6VLX75T-2FF484) using ISE 13.4, both after place&route. The resulting clock frequencies for adders with two and three inputs with 16 to 64 bit output word size are shown in Fig. 2 (top). While both consume the same resources in each architecture, the ternary adders are slower. The relative frequency drop from two to three inputs is shown in Fig. 2 (bottom). While the frequency drop for Stratix IV is only 5% for 16 bit, it is much higher for Virtex 6 (about 40%). Interestingly, the situation becomes worse on Stratix IV and better on Virtex 6 for higher word sizes. A detailed analysis with `trace` revealed that the large drop for Virtex 6 is caused by the external routing (about 0.5 ns) plus one additional LUT delay (about 0.3 ns) which are large compared to the 0.06 ns of 4 bits of carry propagation. For Stratix IV, the internal routing is naturally much faster leading to a smaller frequency drop. It should be noted that although the ternary adders use the same slice/ALM resources they need more routing resources. Utilizing many local routing resources could yield to unusable slice/ALM resources in the same region. It should be more critical for Xilinx than for Altera due to the external carry routing but this has to be investigated in larger designs.

3. PROPOSED OPTIMIZATION ALGORITHM

3.1. Background

The utilization of pipelined adder graphs for MCM is best explained by using an example as given in Fig. 3(a). It realizes a multiplier block with the coefficients from the target set $T = \{7567, 20406\}$. Each node except the input node (value 1) corresponds to a registered adder ('+'), subtractor ('-') or a pure register. Each node also represents an odd integer factor of the input (even factors can be made odd by preceding right shifts which reduces the word size of the adder/subtractor). All edge weights correspond to left shift factors, e. g., node 5 is realized by left shifting the input x by two bits and adding the unshifted input: $2^2x+2^0x = 5x$. The PAG uses nine pipelined operations which are organized in three pipeline stages.

The same multiplier block can be realized with ternary adders as given in Fig. 3(b). Here, only four pipelined operations and two pipeline stages are necessary. The two sign indicators in the nodes correspond to the sign of the 2nd and

3rd input, the first input is positive by definition, e. g., node 10203 is computed by $5 \cdot 2^1 x + 5 \cdot 2^{11} x - 47 \cdot 2^0 x = 10203x$. The operation of each node, a single addition/subtraction of shifted inputs and a shifted output, can be formalized as $\mathcal{A}$-operation [2]. For ternary adders, it is extended to three input arguments:

$$\mathcal{A}_q^3(u, v, w) = |2^{l_u} u + (-1)^{s_v} 2^{l_v} v + (-1)^{s_w} 2^{l_w} w| 2^{-r} \quad (1)$$

The arguments u, v and w correspond to the node inputs and $q = (l_u, l_v, l_w, r, s_v, s_w)$ depicts the configuration, where $l_{u/v/w} \in \mathbb{N}_0$ are left shifts, $r \in \mathbb{N}_0$ is a right shift, and $s_{v/w} \in \{-1, 1\}$ correspond to the signs of v and w.

3.2. Pipeline Depth

The number of pipeline stages can not be lower than the adder depth (AD) of the corresponding adder graph, sometimes referred to as logic depth, which is defined as the number of cascaded adder stages. The minimal AD to realize a multiplication with a constant can be obtained by evaluating the minimum signed digit (MSD) representation of the constant [22]. With a signed digit (SD) representation, each digit is out of the values $\{-1, 0, 1\}$, where digit -1 is usually denoted as $\bar{1}$. The SD number system is redundant, so there are many representations for the same number, e. g., the number 7 can be represented as 111, $100\bar{1}$ or $1\bar{1}00\bar{1}$ (and more). An MSD number is defined to have a minimal number of non-zeros. If an MSD number (the constant) is multiplied by a variable, each non-zero digit corresponds to a partial product. These have to be bit shifted and added (if the non-zero is 1) or subtracted (if the non-zero is $\bar{1}$) to get the final product. Using MSD and an adder tree leads to the minimum adder depth. For two-input adders, each adder stage of the tree can halve the number of partial products. Hence, the minimum adder depth can be computed using

$$\mathrm{AD}^2_{\min}(x) = \lceil \log_2(\mathrm{nz}(x)) \rceil \quad (2)$$

where $\mathrm{nz}(x)$ is the number of non-zero digits of x in MSD representation [22]. For ternary adders, three partial products can be added in each stage, so the minimum adder depth for ternary adder graphs is

$$\mathrm{AD}^3_{\min}(x) = \lceil \log_3(\mathrm{nz}(x)) \rceil = \left\lceil \frac{\log_2(\mathrm{nz}(x))}{\log_2(3)} \right\rceil \quad (3)$$

$$\leq \lceil 0.631 \cdot \mathrm{AD}^2_{\min}(x) \rceil \, . \quad (4)$$

An AD comparison of adder graphs with two and three inputs is shown in Table 1. For $\mathrm{nz}(x) > 4$, using ternary adders always leads to a lower adder depth.

As each extra pipeline stage introduces additional nodes in the PAG, it is unlikely that there exists a graph with higher depth but less cost. Therefore, we define the number of pipeline stages S to be the minimum which is possible from the target coefficient set T:

$$S := S_{\min} = \max_{t \in T} \mathrm{AD}^3_{\min}(t) \quad (5)$$

Table 1: Adder depth comparison for two-input and ternary adder graphs

non-zeros $\mathrm{nz}(x)$	1	2	3	4	5-8	9-16	17-26	27-64
$\mathrm{AD}^2_{\min}(x)$	0	1	2	2	3	4	5	5
$\mathrm{AD}^3_{\min}(x)$	0	1	1	2	2	3	3	4

3.3. Problem Formulation

The most complex part in optimizing a PAG is finding valid intermediate nodes (the u, v, w in (1)). The corresponding configurations can be found easily by evaluating (1). Therefore, the crucial solution of a PAG can be represented by one set X_s for each stage $s = 1 \ldots S$ which contains all node values of pipeline stage s. With this notation, we can formally define the optimization problem:

Definition 1 (Pipelined Ternary Adder MCM Problem) *Given a set of positive target constants $T = \{t_1, \ldots, t_M\}$ and the number of pipeline stages S, find sets $X_{1,\ldots,S-1} \subseteq \{1, 3, 5, \ldots, x_{max}\}$ with minimal cost such that for all $x \in X_s$ for $s = 1, \ldots, S$, there exists an $\mathcal{A}$-configuration q such that $x = \mathcal{A}_q^3(u, v, w)$ with $u, v, w \in X_{s-1}$, $X_0 = \{1\}$ and $X_S = \{odd(t) \mid t \in T\} \setminus \{0\}$, where $odd(t)$ is the absolute value of t divided by 2 until it is odd.*

The upper bound is chosen to $x_{\max} = 2^{B_{\max}+1}$ [2], where $B_{\max}$ is equal to the maximum bit width of the elements in T. The cost metric for FPGA targets will be discussed later.

3.4. Overall Algorithm

The outline of the proposed algorithm RPAGT is given as pseudo code in Listing 1. In comparison to the original RPAG algorithm [11] only the computations of the methods called in lines 2, 8 and 12 are different, the overall algorithm is identical.

First, the number of pipeline stages as well as the elements of the final stage are initialized in lines 2-3. Then, for each pipeline stage, starting from the output stage S, the elements of the preceeding stage $S-1$ are computed. These are called predecessors in the following. For each stage, the elements of the current stage are copied to a working set W and an empty set P is initialized for predecessors (lines 5 and 6). Then, all possible single predecessors are computed from W with `best_single_predecessor()` and the best one, if it exists, is selected in a greedy manner. If no single predecessor was found, up to three predecessors

Listing 1: RPAGT Algorithm

```
1  RPAGT(T)
2    S := max_{t∈T} AD^3_min(t)
3    X_S := {odd(t) | t ∈ T} \ {0}
4    for s = S ... 2
5      W := X_s
6      P := ∅
7      do
8        p ← best_single_predecessor(P, W, s)
9        if p ≠ 0
10         P ← P ∪ {p}
11       else
12         P' ← best_msd_predecessor_set(W, s)
13         P ← P ∪ P'
14       W ← W \ A^3_*(P)
15     while |W| ≠ ∅
16     X_{s-1} ← P
```

are searched by evaluating the partial products obtained by the MSD representations of W with `best_msd_predecessor_set()`. The computation of predecessors is explained in detail in the next subsection. The predecessors found are then inserted in set P and all elements that can be realized from them are now deleted from W (lines 10/13,14). For that, the set $\mathcal{A}^3_*$ is defined, which contains all possible numbers which can be computed from X with a single $\mathcal{A}^3$-operation:

$$\mathcal{A}^3_*(X) = \bigcup_{u,v,w \in X} \left\{ \mathcal{A}^3_q(u,v,w) \mid q \text{ valid configuration} \right\} \quad (6)$$

These steps are repeated until the working set is empty. Then the next lower stage is processed. The algorithm terminates when X_1 is computed (X_0 is known to be $\{1\}$). The termination is guaranteed as predecessors are found in each loop from which at least one element in W can be computed.

As the best local choice in a greedy algorithm does not necessarily leads to the best global solution, a configuration option was implemented to randomly select one of the n'th best predecessors. Using this option, the result can be further enhanced by evaluating several runs of the algorithm.

The cost metric for selecting the best predecessors is defined to be the least predecessors which are able to compute the most elements from the working set, i. e., predecessors P' with the highest gain

$$\text{gain}(P') = \frac{|W'(P')|}{|P'|} \quad (7)$$

with $W'(P') = W \cap \mathcal{A}^3_*(P \cup P')$.

3.5. Predecessor Topologies

To compute the single predecessor elements (function `best_single_predecessor()` in Listing 1), a complete search is performed in which all possible topologies for single predecessors are evaluated. These are the topologies (a)-(f) shown in Fig. 4. In topology (a), a simple register is used, topologies (b-c) use two-input adders and in topologies (d-f) ternary adders are used. The single predecessor topologies are described in the following:

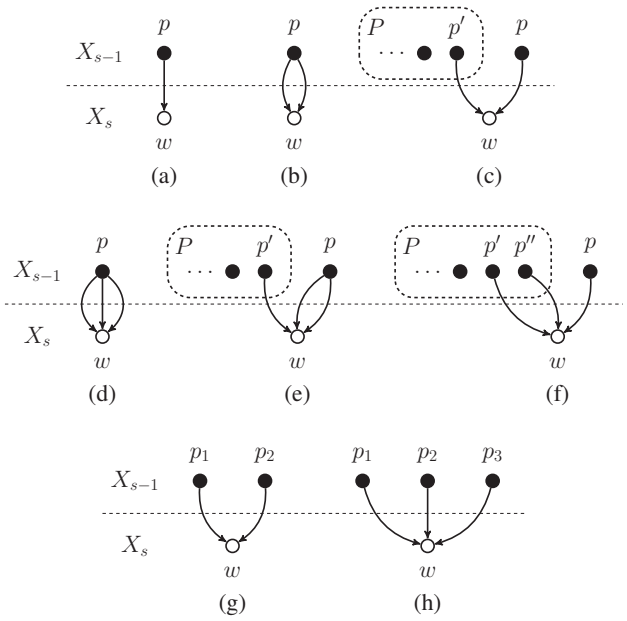

Fig. 4: Predecessor graph topologies for (a)-(f) one, (g) two and (h) three new predecessors

a) Element w has a lower adder depth than s and can be realized by a simple register from p.

b) Element w is computed by $p \cdot (2^k \pm 1)$ with $k \in \mathbb{N}_0$.

c) An already realized element $p' \in P$ is computed with a new predecessor p to get w.

d) Element w is obtained by $p \cdot (2^k \pm 2^l \pm 1)$ with $k, l \in \mathbb{N}_0$.

e) Element $p \cdot (2^k \pm 1)$ is computed with an already realized element $p' \in P$ to get w.

f) Two already realized elements $p', p'' \in P$ are computed with p to obtain w (p' and p'' may be identical).

All operations can be reverted to compute the predecessor p from a given w and p' or p'', if applicable. For that, we define the two-input $\mathcal{A}$-operation $\mathcal{A}^2_q(u,v) := \mathcal{A}^3_q(u,v,0)$. An important property of the $\mathcal{A}$-operation is, that if there exists a configuration q with $w = \mathcal{A}^2_q(u,v)$ then there also exists a configuration q' for which $u = \mathcal{A}^2_{q'}(w,v)$ [2]. It is easy to show that the same property holds for the ternary $\mathcal{A}$-operation. With this property, we can define the rules to compute all possible single predecessors, formally represented by the sets:

$$P_a = \left\{ w \in W \mid \text{AD}^3_{\text{min}}(w) < s \right\} \quad (8)$$

$$P_b = \{w/(2^k \pm 1) \mid \ w \in W,\ k \in \mathbb{N}_0\} \cap \mathbb{N} \setminus \{w\} \tag{9}$$

$$P_c = \bigcup_{\substack{w \in W \\ p' \in P}} \{p = \mathcal{A}_q^2(w, p') \mid q \text{ valid},\ \mathrm{AD}_{\min}^3(p) < s\} \tag{10}$$

$$P_d = \left\{w/(2^k \pm 2^l \pm 1) \mid w \in W,\ k, l \in \mathbb{N}_0\right\} \cap \mathbb{N} \setminus P_a \tag{11}$$

$$P_e = \bigcup_{\substack{w \in W \\ p' \in P}} \left\{p = \mathcal{A}_q^2(w, p')/(2^k \pm 1) \mid q \text{ valid},\ \mathrm{AD}_{\min}^3(p) < s,\ k \in \mathbb{N}_0\right\} \cap \mathbb{N} \setminus P_a \tag{12}$$

$$P_f = \bigcup_{\substack{w \in W \\ p', p'' \in P}} \left\{p = \mathcal{A}_q^3(w, p', p'') \mid q \text{ valid},\ \mathrm{AD}_{\min}^3(p) < s,\ k \in \mathbb{N}_0\right\} \setminus P_a \tag{13}$$

These sets are evaluated and the best predecessor out of all the sets is selected.

If no single predecessor was found, the search continues with evaluating the topologies (g) and (h). This is done in function `best_msd_predecessor_set()` in Listing 1. As there are too many combinations of valid predecessor pairs (g) and triplets (h), an exhaustive search is not feasible for larger coefficients. However, suitable predecessors can be found by combining the non-zeros of their MSD representations [11]. Thus, the search of predecessor pairs of RPAG [11] (topology g) was extended to search for predecessor triplets (topology h). To illustrate this, let us take the number 53 as an example, which has an MSD representation of $53 = 10\overline{1}0101_{\text{MSD}}$, i. e., $53 = 2^6 - 2^4 + 2^2 + 2^0$. Thus, its MSD representation has 4 non-zeros which requires at least $S = 2$ pipeline stages according to (4). A valid predecessor must have a lower adder depth. Thus, it can have up to $3^{S-1} = 3$ non-zeros. Therefore, all permutations of odd triplets of up to three non-zeros are evaluated, which are $P = \{(10\overline{1}, 1, 1), (10001, \overline{1}, 1), (1000001, \overline{1}, 1), (\overline{1}01, 1, 1), (\overline{1}0001, 1, 1), (101, 1, \overline{1}), (10\overline{1}01, 1, 0), (\overline{1}0101, 1, 0)\} = \{(3, 1, 1),\ (17, -1, 1),\ (65, -1, 1),\ (-3, 1, 1),\ (-15, 1, 1),\ (5, 1, -1),\ (13, 1, 0),\ (-11, 1, 0)\}$. Note that triplets containing a zero can be implemented using ordinary two-input adders.

3.6. Example

In this subsection, an example ternary MCM optimization of the coefficient set {7567,20406} is given to illustrate the RPAGT algorithm (Listing 1). The resulting adder graph is the one already discussed earlier and shown in Fig. 3(b).

Both coefficients have 5 non-zeros in their MSD representation which results in a two stage optimization ($S = 2$ in line 2). Then the last stage is initialized to the odd representation of the coefficients ($X_2 = \{7567, 10203\}$) and the working set and predecessor set are initialized to $W = \{7567, 10203\}$ and $P = \emptyset$ (lines 3-6).

In the first iteration of stage 2, the single predecessor topologies (a), (b) and (d) are evaluated as P is empty. Two valid predecessors $p \in \{47, 161\}$ are found to compute 7567 using topology (d). As both have a gain of one (one element from W can be eliminated by a single predecessor), the smaller predecessor 47 is selected and inserted in P while 7567 is eliminated from W. Note that for RPAG with two-input adders no single predecessor would be found, so a pair of predecessors using `best_msd_predecessor_set()` had to be found at this stage.

In the second iteration of stage 2, all single predecessor topologies (a-f) are evaluated with the following valid predecessors to compute the remaining element 10203: $p \in \{5, 59, 1025\}$ for topology (e) using $p' = 47$ and $p \in \{2017, 4093, 32763\}$ for topology (f) using $p' = p'' = 47$. Again, all predecessors have a gain of one and 5 is selected, inserted in P, and 10203 is removed from W.

Now, W is empty which terminates the optimization of the current stage. As all the elements from stage 1 can be directly computed from element 1 of stage 0, the algorithm terminates.

4. RESULTS

To evaluate the proposed algorithm, benchmark coefficients from FIR filter designs were optimized. These were already used in several previous publications [3–5, 11]. Their maximum word size is 16 bit and their numeric values are available online as MIRZAEI10_N, where N is the number of coefficients [23]. The optimization is compared to the state-of-the-art two-input adder optimization of the RPAG algorithm [11] which outperformed previous methods [3,4]. The results are shown Table 2. Due to the symmetry of the filter coefficients and due to equal coefficients after making them odd (step 3 of Listing 1), the number of unique odd coefficients is usually much less than N and is given as N_{uq} in Table 2. For both algorithms, the number of pipeline stages (S), the number of registered add/subtract operations (add ops), the number of pure registers (pure reg.) as well as the sum of both registered operations (reg ops) are given. Clearly, using ternary adders reduces the number of pipeline stages and thereby the latency from $S = 3$ to $S = 2$. As the pipeline registers of adders come for free on FPGAs, the number total registered operations is the most important measure. The improvement of registered operations is shown in the last column of Table 2. On average, about 27% less registered operations are needed by using ternary adders. It can be observed that the improvement is better for smaller coefficient sets. This is due to the fact that for each unique output coefficient N_{uq} there must be at least one add/subtract operation. In addition, at least one add/subtract operation has to be in each of the $S - 1$ non-output stages. This leads to a lower bound of $N_{\text{uq}} + S - 1$ add/subtract operations for pipelined adder graphs, which is identical to the lower bounds derived for non-pipelined adder graphs [24]. Note that S may be less for ternary adders than for two-input

Table 2: Comparison of high level operations between RPAG with two-input adders and the proposed ternary adder optimization RPAGT

		RPAG algorithm [11]				RPAGT (proposed)				
N	N_{uq}	S	add ops	pure reg.	reg. ops	S	add ops	pure reg.	reg. ops	impr.
6	3	3	8	1	9	2	5	0	5	44.4%
10	5	3	10	3	13	2	6	2	8	38.5%
13	7	3	14	2	16	2	8	2	10	37.5%
20	10	3	15	4	18	2	9	3	12	33.3%
28	14	3	20	3	23	2	15	2	17	26.1%
41	21	3	31	1	32	2	23	2	25	21.9%
61	31	3	39	3	42	2	32	2	34	19.0%
119	54	3	62	7	69	2	56	1	57	17.4%
151	71	3	79	4	83	2	72	2	74	10.8%
avg.:	24		30.89	3.56	33.89		25.11	1.78	26.89	27.7%

Table 3: Synthesis results of MCM blocks optimized by RPAG with two-input adders and the proposed ternary adder optimization RPAGT

	Xilinx Virtex 6				Altera Stratix IV			
	RPAG [11]		RPAGT		RPAG [11]		RPAGT	
N	Slices	f_{max} [MHz]	Slices	f_{max} [MHz]	ALMs	f_{max} [MHz]	ALMs	f_{max} [MHz]
6	50	648.1	25	491.4	108	587.2	62	565.0
10	59	671.1	45	484.3	120	588.2	82	548.0
13	82	630.1	49	441.7	160	592.8	125	546.5
20	87	632.1	65	455.0	183	547.7	141	528.5
28	117	601.3	94	470.2	225	545.9	190	534.8
41	165	598.4	129	434.2	314	540.3	264	522.7
61	217	559.0	190	419.6	434	489.7	397	505.6
119	347	520.0	307	383.6	650	462.3	603	469.0
151	432	482.4	407	374.0	768	445.6	787	450.7
avg:	172.9	593.6	145.7	439.3	329.1	533.3	294.6	519.0
imp:	-	-	15.7%	-26.0%	-	-	10.5%	-2.7%

adders. The ternary adder solutions are very close to that lower bound. For larger coefficient sets only a few percent of additional adders are needed. On average, 1.89 registered operations are needed in addition. This indicates that most instances are either optimal or very close to the optimum.

The resulting adder graphs were synthesized for Xilinx and Altera FPGAs to evaluate the improvement in terms of logic resources. For that, a VHDL code generator for pipelined adder graphs including ternary adder was written. The synthesis results are listed in Table 3. They were obtained using the same synthesis tools and FPGAs as described in Section 2.3. The input word size was chosen to 12 bit as in previous publications.

For Altera, the hardware complexity, measured in adaptive logic modules (ALMs) is reduced by 10.5% on average. As expected, the speed is slightly reduced by 2.7%. For Xilinx, the improvement in terms of slices ranges from 5.8% for the largest coefficient set up to 50% for the smallest instance. On average, the slice improvement is 15.7%. The maximum frequency is reduced due to the longer critical path of the ternary adders, but only by 26%. However, the MCM blocks are still very fast. As they are usually part of a more complex system they will most likely not be the limiting factor with only two stages of logic. The average slice and ALM reductions are less than the reduction of registered operations. For Xilinx, this can be explained by the fact that half of the slice flip flops are unused in two-input additions and can be used to implement pure registers without extra resources. This is not the case for ternary adders as the corresponding output is used to compute the carry bit (c_i' in Fig. 1(b)). For Altera, additional ALUTs are needed in particular in larger MCM instances for negating ternary adder inputs as described in Section 2.1.

5. CONCLUSION

An optimization method for the pipelined multiple constant multiplication problem was presented which can take advantage of ternary adder realizations in modern FPGAs. As this method is a generalization of the multiple constant multiplication without pipelining as well as the single constant multiplication (with or without pipelining), it covers a large field of applications. Compared to a state-of-the-art method which uses two-input adders, the number of pipelined operations (pipelined adders/subtractors and registers) could be reduced by 10% to 44%, depending on the benchmark instance. In terms of logic resources, slice reductions of 5.8% to 50% and ALM reductions of up to 42% could be achieved for Xilinx Virtex 6 and Altera Stratix IV FPGAs, respectively. Although the designs are slower compared to two-input adder realizations, they require fewer pipeline stages and are still very fast, achieving clock frequencies of typically more than 370 MHz on Virtex 6 and 450 MHz on Stratix IV FPGAs.

6. 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," *IEEE Asia Pacific Conf. on Circuits and Systems (APCCAS)*, pp. 1555–1558, 2006.

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

[3] 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. 2010, pp. 1–17, 2010.

[4] M. Kumm and P. Zipf, "High Speed Low Complexity FPGA-based FIR Filters Using Pipelined Adder Graphs," in *Int. Conf. on Field-Programmable Technology (FPT)*, 2011, pp. 1–4.

[5] 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 *Proceedings of SPIE*, vol. 8401, 2012.

[6] 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 (DATE)*, 2012, pp. 1197–1202.

[7] A. Crosisier, D. J. Esteban, M. E. Levilio, and V. Riso, "Digital Filter for PCM Encoded Signals," *US Patent No 3777130*, 1973.

[8] S. Zohar, "New Hardware Realizations of Nonrecursive Digital Filters," *IEEE Trans. on Computers*, vol. 22, no. 4, pp. 328–338, 1973.

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

[10] F. Qureshi and O. Gustafsson, "Low-Complexity Reconfigurable Complex Constant Multiplication for FFTs," *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 1137–1140, 2009.

[11] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication," *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 49–52, 2012.

[12] K. Chapman, "Constant Coefficient Multipliers for the XC4000E," *Xilinx Application Note*, 1996.

[13] M. Wirthlin, "Constant Coefficient Multiplication using Look-up Tables," *The Journal of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, 2004.

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

[15] *Stratix II Device Handbook*, Altera Corporation, 2005.

[16] *Virtex-6 FPGA DSP48E1 Slice User Guide UG369 (v1.3)*, Xilinx Inc., 2011.

[17] M. Kumm and P. Zipf, "Hybrid Multiple Constant Multiplication for FPGAs," in *IEEE Int. Conf. on Electronics, Circuits and Systems, (ICECS)*, 2012, pp. 556–559.

[18] RPAG, "Source Code of RPAG and RPAGT," 2013. [Online]. Available: http://www.uni-kassel.de/go/rpag

[19] G. Baeckler, M. Langhammer, J. Schleicher, and R. Yuan, "Logic Cell Supporting Addition of Three Binary Words," *US Patent No 7565388, Altera Coop.*, 2009.

[20] J. M. Simkins and B. D. Philofsky, "Structures and Methods for Implementing Ternary Adders/Subtractors in Programmable Logic Devices," *US Patent No 7274211, Xilinx Inc.*, Mar. 2006.

[21] OpenCores, "Ternary Adder IP Core," 2013. [Online]. Available: http://opencores.org/project,ternary_adder

[22] O. Gustafsson, A. Dempster, K. Johansson, M. Macleod, and L. Wanhammar, "Simplified Design of Constant Coefficient Multipliers," *Circuits, Systems, and Signal Processing*, vol. 25, no. 2, pp. 225–251, 2006.

[23] FIRsuite, "Suite of Constant Coefficient FIR Filters," 2013. [Online]. Available: http://www.firsuite.net

[24] O. Gustafsson, "Lower Bounds for Constant Multiplication Problems," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 54, no. 11, pp. 974–978, Nov. 2007.