Constant Matrix Multiplication with Ternary Adders

Martin Hardieck, Martin Kumm, Patrick Sittel and Peter Zipf
University of Kassel, Digital Technology Group, Germany
{hardieck,kumm,sittel,zipf}@uni-kassel.de

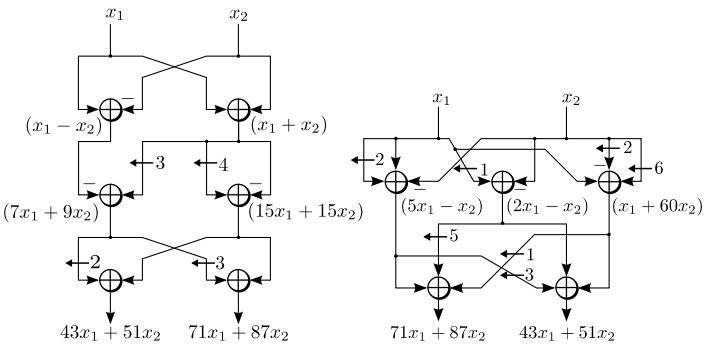
***Abstract*—Constant matrix multiplication (CMM), i. e., the multiplication of a constant matrix with a vector, is a common operation in digital signal processing. The CMM operation can be realized multiplierless using only additions/subtractions and bit shifts. Modern FPGAs support the efficient mapping of ternary adders, i. e., adders with three inputs. Previous work has shown that the usage of ternary adders for the FPGA implementation of other multiplierless constant multiplication problems is very beneficial. However, no algorithm exists to optimize CMM operations with ternary adders. This work proposes a novel heuristic approach to further reduce CMM complexity for FPGAs by exploiting ternary adders. The algorithm can be targeted for combinatorial CMM with minimal depth or fully pipelined CMM operations. It is shown experimentally that 30% less operations are needed on average by using ternary adders, resulting in 11% LUT reductions.**

I. Introduction

A frequently used operation in digital signal and image processing is the multiplication of a vector with a constant matrix, commonly referred to as constant matrix multiplication (CMM). Examples are discrete transforms like the fast Fourier transform (FFT), the discrete cosine transform (DCT), color space conversion in image and video processing, polyphase filters as used in sample rate conversions, and error correcting codes [1]. Those applications perform a large number of multiplications with constants. At design time, it is possible to avoid the costly multipliers using only additions, subtractions and bit-shifts for this task [1]–[6].

Finding a multiplierless solution with minimum number of additions and subtractions is an optimization problem known as the CMM problem. It is a generalization of the multiple constant multiplication (MCM) problem [7], [8] for which only one input is processed (instead of an input vector). MCM itself is a generalization of the single constant multiplication (SCM) problem where a single input is multiplied by a single constant.

The SCM problem was shown to be an NP-hard optimization problem [9], so it follows that the CMM problem also belongs to this class of intractable problems. Several heuristic solutions for the CMM problem were proposed [1], [3], [4], [10], [11]. These formulations use the common two-input adders to realize a CMM operation. However, using two-input adders to map CMM circuits to modern FPGAs is not the most efficient way as the used ripple carry adders (RCAs) typically underutilize the resources of 6-input LUTs in conjunction with fast carry chains. For a better hardware utilization, so-called *ternary adders*, i. e., adders which are able to add three numbers, offer a much more efficient realization. A ternary adder realizes a summation of the form $s = x + y + z$ with the same resources as a common two-input adder but at a slightly lower speed.



(a) CMM with two-input adders (b) CMM with ternary adders

Fig. 1. Example CMM operations

In our previous work, we could show that using ternary adders for SCM [12] and MCM [13] reduces the operation count by about one third on average. This translates to significant resource reductions on Intel and Xilinx FPGAs. So far, no algorithm exists for optimizing the generalized CMM problem using ternary adders. This work closes this gap by proposing a CMM optimization which is aware of ternary adders, thus, further improving the state-of-the-art in CMM targeting FPGAs.

A multiplierless CMM operation can be represented as an *adder graph*. An adder graph is a directed acyclic graph (DAG) where each node corresponds to an adder or subtractor and edge weights correspond to bit shifts. The bit shifts are usually assumed without cost as they are hard-wired. An illustrating example of a CMM circuit for the operation

$$\begin{pmatrix} 43 & 51 \\ 71 & 87 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 43x_1 + 51x_2 \\ 71x_1 + 87x_2 \end{pmatrix} \quad (1)$$

is given in Fig. 1. The horizontal arrows denote left shifts and each add/subtract operation corresponds to a sum-of-product which is denoted in parenthesis. The solution using two-input add/subtract nodes is shown in Fig. 1(a) and was obtained by the CMM heuristic RPAG-CMM, recently proposed in [1]. A solution using add/subtract nodes with ternary adders is shown in Fig. 1(b). Even for this small instance, a reduction of add/subtract operations from six to five can be observed. Additionally, the *adder depth* (AD), which is defined as the number of cascaded adder stages, could be reduced from three to two, improving delay or latency (in a pipelined implementation) by about one third.

II. Proposed Algorithm

The proposed CMM algorithm for ternary adders is an extension of our recently proposed CCM algorithm [1], called RPAG-CMM, combined with ideas from our MCM algorithm for ternary adders RPAGT [13]. In the following, we start with some preliminaries about the formal adder graph representation, the computation of the AD and we will then define the core problem and the algorithm.

A. Adder Graph Representation

Each node in a CMM adder graph corresponds to a vector of inputs [14] which in turn represents the sum of weighted inputs. Element i of this vector corresponds to the (multiplicative) weight of input i, e. g., the computation of $43x_1 + 51x_2$ in the leftmost output of Fig. 1(a) is represented by vector $(43, 51)$. Each input is hence represented by its corresponding unit vector. Each node in the adder graph performs an addition of up to three possibly bit-shifted inputs where up to two inputs may be negated at the same time. This operation is commonly described as an $\mathcal{A}$-operation which is here extended for the ternary CMM case to

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

The arguments $\vec{u}$, $\vec{v}$ and $\vec{w}$ 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 $s_{v/w} \in \{0, 1\}$ correspond to the signs of $\vec{v}$ and $\vec{w}$. Note that all shift operations are performed element-wise.

B. Adder Depth and Pipeline Depth

One important property of an adder graph is its AD. The minimal necessary AD for a node can be obtained by evaluating the canonic signed digit (CSD) representation of its vector elements [15]. Each non-zero digit in the CSD corresponds to a partial product which has to be bit-shifted and added/subtracted to get the final product. For each layer of ternary adders, the number of non-zero digits can be reduced by a factor of three, leading to a minimum depth of

$$\mathrm{AD}_{\min}(\vec{c}) = \left\lceil \log_3 \left(\sum_{n=1}^{N} \mathrm{nz}(c_n) \right) \right\rceil \quad (3)$$

where $\mathrm{nz}(c_n)$ denotes the minimum number of non-zero digits of element c_n. The total AD for the complete adder graph can be obtained from the maximum AD of the rows of the constant matrix $\mathbf{C}$ [15]. If the $N \times M$ matrix $\mathbf{C}$ is defined as a column vector with size M of row vectors with size N each, i. e., $\mathbf{C} = (\vec{c}_1 \vec{c}_2 \dots \vec{c}_M)^T$ the total AD is

$$\mathrm{AD}_{\min}(\mathbf{C}) = \max_m \mathrm{AD}_{\min}(\vec{c}_m) \ . \quad (4)$$

Each ternary adder can be pipelined by using the optional slice flip flops without additional resources. Hence, to obtain the highest throughput it is necessary to put a pipeline register after each ternary adder. In this case, the minimum number of pipeline stages is limited by the AD, i. e., $S := \mathrm{AD}_{\min}(\mathbf{C})$.

C. Core CMM Problem

To reduce the number of different vector representations, all vectors are normalized (denoted by $\mathrm{norm}(\vec{c}_m)$) by the element-wise division by two until at least one of the elements is odd [3]. This step can later be reversed by left shifting the vector.

The problem in finding a minimum adder graph can be reduced to the core problem of finding the normalized node vectors of that adder graph. The remaining bit-shifts can be found easily by evaluating (2). Therefore, we follow the same approach as in [1] to define a set X_s for each adder stage s. Set X_0 contains the M different input unit vectors. The initialization of the remaining X_s sets determines the objective ('pipelining' or 'minimal AD'). For a pipelined CMM optimization, the last stage X_S is initialized to the normalized row vectors of matrix $\mathbf{C}$

$$X_S = \{\mathrm{norm}(\vec{c}_m)\} \text{ for } m = 1 \dots M \quad (5)$$

while for the 'minimal AD' target, each vector is put into the minimum possible stage

$$X_s = \{\mathrm{norm}(\vec{c}_m) \mid \mathrm{AD}_{\min}(\vec{c}_m) = s\} \text{ for } m = 1 \dots M \ . \quad (6)$$

The CMM problem can now be stated as follows: Given the initial sets $X_{1,\dots,S-1}$, add the vectors with least total cost such that for all $\vec{x} \in X_s$ for $s = 1, \dots, S$, there exists an $\mathcal{A}$-configuration q such that $\vec{x} = \mathcal{A}_q(\vec{u}, \vec{v}, \vec{w})$ with $\vec{u}, \vec{v}, \vec{w} \in X_{s-1}$.

D. Optimization Algorithm

The overall optimization algorithm is identical to our previous RPAG-CMM algorithm [1]. The contribution here is the use of the generalized definitions for the $\mathcal{A}$-operation, the AD and the initialization with the X_s sets for the ternary CMM case as described above. In addition, a key part is the evaluation of the vectors extended to ternary adders which is described in Section II-E. The depth search and the overall search are briefly described in the following for completeness but more details can be found in [1].

1) Depth Search: The overall algorithm uses a depth-first search in each iteration. Each depth-first search starts from the output stage $s = S$ and searches for vectors in stage $s - 1$ for which the elements in the current stage s can be computed. Hence, set X_{s-1} is filled with vectors (if any) until set X_s can be computed from that set. For that, it is first checked if any element from X_s can be directly computed from X_{s-1}. For the remaining elements of X_s, predecessors are searched from which as many elements of X_s as possible can be computed when added to X_{s-1}. The predecessor search starts by evaluating single predecessors and continues with multiple predecessors (up to three for ternary adders), for which it is guaranteed to find a valid solution (for details, see Section II-E). When X_{s-1} is complete such that each element from X_s can be computed by a (two-input or ternary) addition, the next lower stage is optimized by decrementing s until stage $s = 1$ is reached.

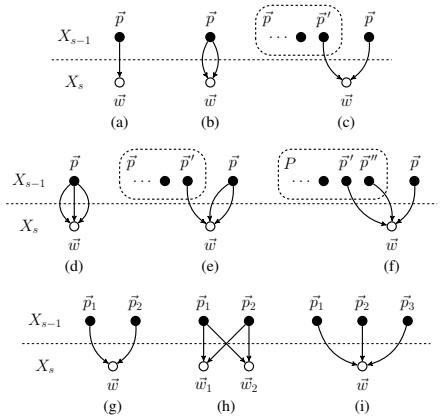


Fig. 2. Predecessor graph topologies for (a)-(f) one, (g-h) two and (i) three new predecessors

2) Overall Algorithm: As the evaluation of predecessor vectors can only be done from a local point of view, several depth-first searches are performed in the overall algorithm. During the first depth-first search, the locally best option is selected which corresponds to a greedy search. In the following iterations, the 2nd best, 3rd best up to the L'th best options of the first decision are evaluated. Next, the best result for the first decision is fixed and the next decision is evaluated in the same manner until all decisions are evaluated. This can be interpreted as a greedy search (overall algorithm) on top of another greedy search (the depth search of Section II-D1).

E. Predecessor Computation

One key in the algorithm is the search for valid predecessors. For that, the topologies (a)-(f) shown in Fig. 2 are evaluated. In topology (a), a simple wire (combinatorial) or register (pipelined) is used, topologies (b-c) use two-input adders and in topologies (d-f) ternary adders are used. The single predecessors can be computed from these topologies as follows:

$$P_a = \left\{\vec{w} \in X_s \mid \mathrm{AD}^3_{\min}(\vec{w}) < s\right\} \tag{7}$$

$$P_b = \{\vec{w}/(2^k \pm 1) \mid \ \vec{w} \in X_s,\ k \in \mathbb{N}_0\} \cap \mathbb{N} \setminus \{\vec{w}\} \tag{8}$$

$$P_c = \bigcup_{\substack{\vec{w} \in X_s \\ \vec{p}\,' \in X_{s-1}}} \{\vec{p} = \mathcal{A}^2_q(\vec{w}, \vec{p}\,') \mid q \text{ valid},\ \mathrm{AD}^3_{\min}(\vec{p}) < s\} \tag{9}$$

$$P_d = \left\{\vec{w}/(2^k \pm 2^l \pm 1) \mid \vec{w} \in X_s,\ k, l \in \mathbb{N}_0\right\} \cap \mathbb{N} \setminus P_a \tag{10}$$

$$P_e = \bigcup_{\substack{\vec{w} \in X_s \\ \vec{p}\,' \in X_{s-1}}} \left\{\vec{p} = \mathcal{A}^2_q(\vec{w}, \vec{p}\,')/(2^k \pm 1) \mid q \text{ valid},\ \mathrm{AD}^3_{\min}(\vec{p}) < s,\ k \in \mathbb{N}_0\right\} \cap \mathbb{N} \setminus P_a \tag{11}$$

$$P_f = \bigcup_{\substack{\vec{w} \in X_s \\ \vec{p}\,', \vec{p}\,'' \in X_{s-1}}} \left\{\vec{p} = \mathcal{A}^3_q(\vec{w}, \vec{p}\,', \vec{p}\,'') \mid q \text{ valid},\ \mathrm{AD}^3_{\min}(\vec{p}) < s,\ k \in \mathbb{N}_0\right\} \setminus P_a \tag{12}$$

These sets are evaluated and the best predecessor out of all the sets (i. e., the one that can eliminate most vectors from X_s) is selected. In case no single predecessor was found, the topologies (g)–(i) are evaluated. The computation of predecessors for topologies (g) and (h) is identical to the one proposed in [1] while the evaluation of topology (i) is a straight forward extension of topology (g): The non-zero digits of the vectors in $\vec{w}$ are distributed into three predecessors which allows a lower AD than topology (g).

TABLE I
COMPARISON OF DIFFERENT CMM METHODS FOR BENCHMARK MATRICES FROM THE LITERATURE (BEST MARKED BOLD)

Matrix	Size	Method	Adders	adder depth (AD)	Registered Ops
$\mathbf{C}_1$	3×3	[1]	12	4	16
		proposed	**10**	**3**	**13**
$\mathbf{C}_2$	3×3	[3], [16]	11	5	22
		[1]	12	4	16
		proposed	**7**	**3**	**10**
$\mathbf{C}_3$	4×4	[1]	14	4	19
		proposed	**9**	**3**	**13**
$\mathbf{C}_4$	2×2	[1]	6	**3**	7
		proposed	**3**	**3**	**5**
$\mathbf{C}_5$	2×2	[1]	5	**2**	5
		proposed	**3**	**2**	**4**
$\mathbf{C}_6$	4×4	[1], [17]	8	**2**	**8**
		proposed	**6**	**2**	**8**
$\mathbf{C}_7$	8×8	[1]	20	4	20
		proposed	**18**	**2**	**18**
$\mathbf{C}_8$	16×16	[1]	31	**3**	47
		proposed	**26**	**3**	**40**

III. RESULTS

In the following, the proposed algorithm for the CMM problem with ternary adders is evaluated and compared against state-of-the-art approaches by their operation count and logic resources. This is done using benchmark matrices of realistic applications from literature that were also used in [1]. Matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ are color space conversion matrices for RGB to YCbCr and vice versa. Matrices $\mathbf{C}_3$-$\mathbf{C}_5$ are numeric examples taken from Figure 4 of [3], Figure 2 of [11] and Figure 1 of [18], respectively. Matrix $\mathbf{C}_6$ is used in H.264/AVC video coding and its CMM solution is taken from Figure 1 of [17]. Matrices $\mathbf{C}_7$ and $\mathbf{C}_8$ are used in an 8×8 Hadamard transform and a (16,11) Reed-Muller error correcting code.

Table I shows the results from optimization, such as the adder count, the adder depth and the number of registered operations (sum of pipelined adders and registers for a balanced pipeline) for the proposed method and the best result(s) reported previously [1]. It can be observed that the proposed CMM optimization using ternary adders finds a lower operation count in all examined cases compared to previous methods, leading to an average adder and registered operation reduction of 30% and 24%, respectively. Also, the achieved AD is always lower or equal compared to the state-of-the-art.

To examine that the operation count also leads to a lower FPGA resource usage, synthesis experiments have been performed. For that, a VHDL code generator was developed with the help of FloPoCo [19]. The synthesis results (after place&route) were obtained for an input word size of 12 bit mapped to a Xilinx Virtex 6 FPGA (XC6VLX75T-FF784-2) using ISE 13.4 to allow a direct comparison to previous

TABLE II
SYNTHESIS RESULTS FOR THE DESIGNS OF TABLE I

Matrix	Method	comb. CMM (min. AD) LUTs	DSPs	f_{max} [MHz]	pip. CMM LUTs	DSPs	f_{max} [MHz]
C1	baseline	379	3	192.1	415	3	245.2
	ref [1]	189	0	256.5	222	0	619.2
	proposed	188	0	175.4	227	0	436.9
C2	baseline	109	3	214.7	119	3	255.8
	ref [1]	190	0	215.5	224	0	592.8
	ref [16]	186	0	218.4	202	0	702.3
	proposed	213	0	172.2	192	0	436.1
C3	baseline	397	0	204.9	427	0	404.5
	ref [1]	228	0	244.6	257	0	648.9
	proposed	160	0	184.7	177	0	456.4
C4	baseline	177	0	204.9	176	0	331.2
	ref [1]	88	0	306.2	92	0	651.9
	proposed	54	0	263.0	70	0	474.2
C5	baseline	136	0	333.3	113	0	462.1
	ref [1]	72	0	494.3	71	0	651.9
	proposed	49	0	372.2	66	0	495.8
C6	baseline	153	0	278.2	202	0	636.5
	ref [1]	111	0	424.1	108	0	704.2
	proposed	91	0	258.3	109	0	471.5
C7	baseline	561	0	217.2	854	0	610.9
	ref [1]	284	0	251.8	294	0	484.5
	proposed	286	0	241.9	281	0	380.5
C8	baseline	414	0	223.2	839	0	578.7
	ref [1]	424	0	200.0	532	0	513.6
	proposed	397	0	144.1	568	0	351.1
Avg.	baseline	290.8	0.75	233.6	393.1	0.75	440.6
	ref	198.3	0	299.1	225.0	0	608.4
	proposed	176.4	0	232.2	212.5	0	471.1
Impr.	baseline	39.3%		−0.6%	45.9%		6.5%
	ref	11.0%		−28.8%	5.6%		−29.1%

results [1]. The synthesis results are listed in Table II. For most cases, a resource reduction can be observed. The average LUT reduction is 11% for the combinatorial CMM and 5.6% for the pipelined CMM compared to the best known multiplierless design, respectively. The resource reduction is less than the operation count reduction obtained above. This can be explained by the typically larger word sizes used in the ternary adders. For example, in the worst result of C8 (pipelined), the LUT count is even increased from 532 to 568 (+6.8%). Changing the input word size from 12 to 32 bit reduces this effect leading to a LUT reduction from 1546 to 1388 (−10.2%). As ternary adders are known to be slower than two-input adders, this resource reduction comes at the price of a slightly reduced clock frequency. However, it is unlikely that a CMM block will be part of the critical path when embedded in a more complex design.

IV. CONCLUSION

A method to solve multiplierless constant matrix multiplication problems using ternary adders was presented for the first time. It was shown that significant resource and LUT reductions are possible compared to common two-input adder implementations. The implementation of the optimization algorithm is available as open-source within the PAGSuite project [20].

REFERENCES

[1] M. Kumm, M. Hardieck, and P. Zipf, "Optimization of Constant Matrix Multiplication with Low Power and High Throughput," *IEEE Transactions on Computers*, vol. 66, no. 12, pp. 2072–2080, 2017.

[2] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–577, 1995.

[3] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Low-Complexity Constant Coefficient Matrix Multiplication Using a Minimum Spanning Tree Approach," in *Proceedings of the 6th Nordic Signal Processing Symposium (NORSIG)*, 2004, pp. 141–144.

[4] M. D. Macleod and A. Dempster, "Common Subexpression Elimination Algorithm for Low-Cost Multiplierless Implementation of Matrix Multipliers," *Electronics Letters*, vol. 40, no. 11, pp. 651–652, 2004.

[5] A. Kinane, V. Muresan, and N. O'Connor, "Towards an Optimised VLSI Design Algorithm for the Constant Matrix Multiplication Problem," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2006, pp. 5111–5114.

[6] ——, "Optimisation of Constant Matrix Multiplication Operation Hardware Using a Genetic Algorithm," in *Lecture Notes in Computer Science*. Springer, 2006, pp. 296–307.

[7] D. R. Bull and D. H. Horrocks, "Primitive Operator Digital Filters," *IEE Proceedings of Circuits, Devices and Systems*, vol. 138, no. 3, pp. 401–412, Jun. 1991.

[8] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, pp. 1–38, 2007.

[9] P. Cappello and K. Steiglitz, "Some Complexity Issues in Digital Signal Processing," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, Oct. 1984.

[10] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A New Algorithm for Elimination of Common Subexpressions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 18, no. 1, pp. 58–68, 1999.

[11] A. Dempster, O. Gustafsson, and J. O. Coleman, "Towards an Algorithm for Matrix Multiplier Blocks," in *IEEE European Conference on Circuit Theory and Design (ECCTD)*, 2003, pp. 1–4.

[12] M. Kumm, O. Gustafsson, M. Garrido, and P. Zipf, "Optimal Single Constant Multiplication using Ternary Adders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 7, pp. 928–932, 2018.

[13] M. Kumm, M. Hardieck, J. Willkomm, P. Zipf, and U. Meyer-Baese, "Multiple Constant Multiplication with Ternary Adders," in *IEEE International Conference on Field Programmable Logic and Application (FPL)*, 2013, pp. 1–8.

[14] O. Gustafsson and L. Wanhammar, "A Novel Approach to Multiple Constant Multiplication Using Minimum Spanning Trees," *IEEE Midwest Symposium on Circuits and Systems (MWSCAS)*, vol. 3, 2002.

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

[16] K. Holm and O. Gustafsson, "Low-Complexity and Low-Power Color Space Conversion for Digital Video," in *Norchip Conference*. IEEE, 2006, pp. 179–182.

[17] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 598–603, 2003.

[18] S. Ghissoni, E. Costa, J. Monteiro, and R. Reis, "Combination of Constant Matrix Multiplication and Gate-Level Approaches for Area and Power Efficient Hybrid Radix-2 DIT FFT Realization," in *IEEE International Conference on Electronics, Circuits and Systems, (ICECS)*, 2011, pp. 567–570.

[19] F. de Dinechin and B. Pasca, "Custom Arithmetic Datapath Design for FPGAs using the FloPoCo Core Generator," *IEEE Design & Test of Computers*, no. 99, pp. 1–1, 2012.

[20] M. Kumm, K. Möller, M. Hardieck, and P. Zipf. (2018) PAGSuite project website. [Online]. Available: http://www.uni-kassel.de/go/pagsuite