High Speed Low Complexity FPGA-based FIR Filters Using Pipelined Adder Graphs

Martin Kumm and Peter Zipf
Digital Technology Group
University of Kassel, 34121 Kassel, Germany
Email: {kumm, zipf}@uni-kassel.de

***Abstract*—A method for generating high speed FIR filters with low complexity for FPGAs is presented. The realization is split into two parts. First, an adder graph is obtained using an existing multiple constant multiplication (MCM) algorithm. This adder graph describes the required multiplier block of the FIR filter using only additions/subtractions and shifts. Secondly, a novel FPGA-specific combined schedule and pipeline optimization is performed to gain the maximum speed while using a minimal performance penalty. FPGA-specific characteristics are exploited during optimization including the reduction of pipeline registers by duplicating adders in later stages. The optimization is formulated as binary integer linear programming (BILP) problem. It is shown that the generated number of pipelined operations based on the H_{cub} MCM algorithm is reduced up to 29.1% on average compared to an as-soon-as-possible (ASAP) scheduling using cut-set retiming. Synthesis results are obtained by generating VHDL code, showing that the proposed method outperforms the recently proposed Add/Shift method in resource complexity (54.1% reduction on average) while a competitive performance is achieved (88.2% speed of Add/Shift on average).**

I. Introduction

The finite impulse response (FIR) filter is one of the key applications in digital signal processing (DSP). It is used in many DSP systems due to its linear-phase, strict stability, and high-throughput. Nowadays, field programmable gate arrays (FPGAs) are often used for filter applications as they offer the processing of real time signals up to several MHz while providing more flexibility than application specific integrated circuits (ASICs).

A lot of research has been done so far on the hardware efficient implementation of FIR filters. The structure of an FIR filter in transposed form is shown in Fig. 1. This structure is beneficial compared to the direct form as the delay elements (z^{-1}) can be used as pipeline registers for the structural adders (adders outside the dashed box). In ASIC designs, the multiplication by several constants is usually reduced to additions, subtractions and shifts. Finding the minimal configuration of adders[1] (shifts are assumed to be free) is known as multiple constant multiplication (MCM) problem [1]–[6]. The optimization problem is NP-complete [6] and has been an active research topic for almost the last two decades.

MCM algorithms can be divided into common-subexpression elimination (CSE) [1]–[3] and graph based methods [4]–[6]. CSE methods try to identify common patterns in the coefficients while graph based methods try to construct the coefficient values in a bottom-up manner. Graph based methods are more complex (in runtime) but are independent of the number representation [3] and usually find better solutions than CSE-based methods [6]. One of the leading MCM heuristics in terms of quality of optimization results and runtime is the H_{cub} algorithm [6] which outperforms the previous graph based methods BH [4], BHM and RAG-n [5]. The implementation of H_{cub} is open source [7]. In a recent version, the total adder depth (AD) can be limited, which is the maximum number of adder stages from the input to all outputs. This leads to a shorter critical path while the amount of adders is slightly increased.

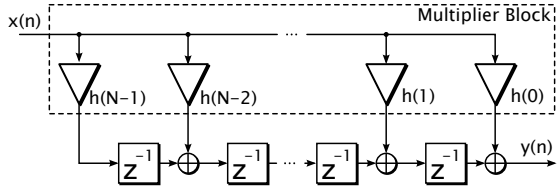

Fig. 1. FIR filter in transposed form

For efficient FPGA-based realizations of FIR filters the features of the architecture must be considered. These are in particular arithmetic components like full adders and embedded multipliers as well as look-up tables (LUTs) and flip-flops (FF). Distributed arithmetic (DA) has long been the favored method to implement FIR filters, as LUTs and adders are used, which perfectly matches the FPGA architecture. However, Meyer-Baese [8] et al. showed that an FPGA-based parallel DA implementation needs 71% more resources on average than a pipelined adder graph realization computed by the RAG-n algorithm [5]. Similar results were also achieved by the Add/Shift method of Mirzaei et al. [2], where a CSE method with FPGA-specific cost function was used. A LUT and FF reduction of 58.7% and 25% could be achieved, respectively, compared to the parallel DA implementation. The performance of both methods was almost the same. It was pointed out in [2], that adder graphs of H_{cub} on the one hand lead to a much lower resource usage compared to their CSE method (72% LUT and 11% FF reduction) but on the other hand lead to a poor performance (68% drop), as pipelining and the use of FPGA-specific features are not considered in the algorithm.

[1]Adders and subtractors will be referred as adders from now on, as both need approximately the same resources.

978-1-4577-1740-6/11/$26.00

In this work, a novel method to produce pipelined multiplier blocks with low resource usage is presented. It is based on adder graphs, e. g. obtained by an MCM algorithm, which are further optimized with FPGA-specific scheduling and retiming.

II. Pipelining of Adder Graphs

A. Pipelining using Cut-Set Retiming

The starting point for retiming is an adder graph obtained by an MCM algorithm. An example adder graph is shown in Fig. 2(a), which realizes a multiplier block with constants of the coefficient set $C = \{480, 512, 846, 1020\}$. This example is used throughout this paper to illustrate the algorithms. The graph needs four adders and was computed using $\mathrm{H}_{\mathrm{cub}}$.

All nodes drawn as circle except the input node (value 1) correspond to adders or subtractors. The value of each node is equal to the factor that is realized at this node, output nodes are drawn without circle. All edge weights are shift factors, e. g., node 15 is realized by shifting the input x by 4 bit and subtracting the unshifted input: $15x = 2^4x - 2^0x$.

Pipelining of directed acyclic graphs like adder graphs can be easily performed using cut-set retiming (CSR) [9]. The realization of the most complex coefficient in the example has an AD of three (node 423). Therefore, the number of pipeline stages is set to three (without input/output registers). One possible pipelined adder structure after CSR is shown in Fig. 2(b). Each node drawn as box includes a register, i. e. stands for either a pure register (one input) or a registered adder (two inputs). Assuming the same computation time for each addition, the maximum clock frequency of the structure in Fig. 2(b) can be approximately three times higher than for the original structure if inputs and outputs are registered. In total 11 additional registers are used to pipeline the graph, which is a huge expense compared to the four adders of the structure without pipelining.

Taking a closer look at the pipelined structure it reveals that by using a different scheduling, some of the registers can be eliminated. For example, coefficient 255 depends on input node 1, which is also available in pipeline stage 2, so it can be moved to stage 3 saving two registers. Further registers can be saved by utilizing FPGA characteristics, which are discussed in the following.

B. Pipelining for FPGA-Architectures

The basic logic elements used in modern FPGAs consist at least of a look-up table (LUT), full-adder (FA) logic with fast carry chain and an optional output flip-flop (FF). These elements are common to all modern FPGAs even if FPGA manufacturers organize them in different units. The unit containing one LUT, FA and FF is referred as basic logic element (BLE) in the following.

Adders are usually implemented as ripple-carry adders on FPGAs requiring one FA or one BLE for each bit. Hence, an N bit adder or subtractor needs N BLEs. Furthermore, the number of BLEs needed to implement an N bit adder is equal to the implementation of an N bit registered adder. Thus, the pipelining of single adders can be realized without extra costs. A pure N bit register needs N BLEs.

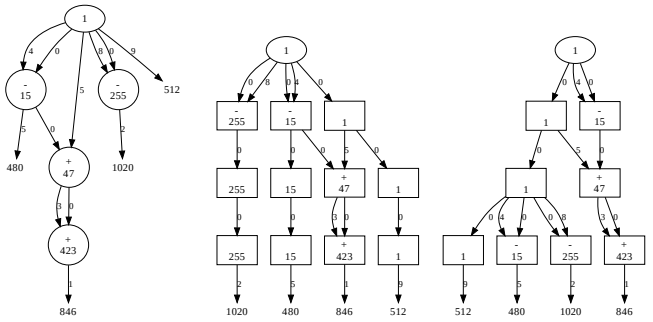

Fig. 2. Adder graph for the coefficient set $\{480, 512, 846, 1020\}$ (a), the pipelined adder graph after cut-set retiming (b), the FPGA optimized pipelined adder graph (c)

Using this model and assuming equal word sizes on each edge for simplification, it becomes obvious that each node drawn as box in Fig. 2(b) requires approximately the same hardware resources. Therefore, the register for factor 15 in pipeline stage 2 of Fig. 2(b) can be eliminated if the coefficient is computed again in the last stage. This optimization step is called *adder duplication* in the following. The optimized pipelined adder graph is shown in Fig. 2(c). Three registered operations could be saved in total compared to Fig. 2(b).

III. Operation Minimized Pipelining using Binary Integer Linear Programming

The procedure in finding the best scheduling as well as the best adder duplication is in general not such a trivial task as in the example above. For example, eliminating a register in one pipeline stage may exclude the elimination of several other registers. The optimization problem of finding the pipelined adder graph with minimal number of nodes can be formulated as binary integer linear programming (BILP) problem which is derived in the following.

The pipelined adder graph is described using a binary $P \times K$ scheduling matrix $\mathbf{S}$. Element $s_{p,k}$ of $\mathbf{S}$ is '1' if constant c_k is scheduled in pipeline stage p. K and P denote the number of unique odd coefficients (including non-output coefficients) and the number of pipeline stages, respectively. P is equal to the adder depth of the adder graph. All even coefficients can be realized by wired shifts from odd coefficients.

The scheduling matrix of the graph in Fig. 2(c) is given by

$$\mathbf{S} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{1}$$

where the columns correspond to the sorted odd coefficients $c_1 = 1$, $c_2 = 15$, $c_3 = 47$, $c_4 = 255$ and $c_5 = 423$.

The optimization problem can now be formulated as minimizing the linear objective function

$$O(\mathbf{S}) = \sum_{p=1}^{P} \sum_{k=1}^{K} s_{p,k} \quad , \tag{2}$$

i. e., minimizing the number of ones in **S**, while considering additional design constraints.

For each coefficient and pipeline stage the constraints are formulated using equalities and inequalities to incorporate all dependencies. The adder input values to compute the actual coefficient c_k are named c_i and c_j. Depending on the coefficient value and the input nodes, there are three cases that lead to different equalities/inequalities:

i) $c_k = 1$: A 1 can be realized in stage 1 and does not need to be computed. To produce a 1 at stage p, a 1 must be available at stage $p-1$. For all $p > 1$, the following inequality must hold:

$$s_{p,k} - s_{p-1,k} \leq 0 \tag{3}$$

ii) $c_k \neq 1$, $c_i = c_j$: Node c_k depends on a single node (like 15, 255 or 423 in the example), so it can only be realized in stage p if either node c_i (or c_j) is available in stage $p-1$ or c_k already exists in stage $p-1$. Furthermore, if $c_i \neq 1$ (like for $c_k = 423$), it can not be realized in stage 1, leading to the following equalities/inequalities:

$$\begin{aligned} s_{p,k} &= 0 \quad | \quad \text{for } c_i \neq 1 \text{ and } p = 1 \\ s_{p,k} - s_{p-1,i} - s_{p-1,k} &\leq 0 \quad | \quad \text{for } p > 1 \end{aligned} \tag{4}$$

iii) $c_k \neq 1$, $c_i \neq c_j$: This case covers nodes that depend on two preceding nodes c_i and c_j (like 47 in the example that depends on 1 and 15). They can only be realized in stage p if either node c_i and c_j are both available in stage $p-1$ or c_k already exists in stage $p-1$. The realization in stage 1 is in general not possible, which results in

$$\begin{aligned} s_{p,k} &= 0 \quad | \text{ for } \quad p = 1 \\ 2s_{p,k} - s_{p-1,i} - s_{p-1,j} - 2s_{p-1,k} &\leq 0 \quad | \text{ for } \quad p > 1 \end{aligned} \tag{5}$$

Of course, all output coefficients must be available in the last pipeline stage leading to the constraint:

$$s_{P,k} = 1 \tag{6}$$

IV. Results

A. Results from Optimal Pipelining

The following experiments were done using the filter benchmark set of [2], which contains FIR filters from 6 to 151 taps. The coefficient data are available online in the FIR benchmark suite [10] labeled as MIRZAEI10_N, where N denotes the number of coefficients. Adder graphs were obtained using the H_{cub} algorithm which was configured either for minimal adder count or minimal adder depth, which is referred as 'H_{cub} AD min.' in the following. The adder depth is critical for pipelining, as it directly determines the number of pipeline stages P. Hence, even if more adders are needed for a lower adder depth, pipeline registers may be saved due to a lower number of pipeline stages.

The BILP problem of the schedule and pipeline optimization was solved using the Matlab `bintprog` method. For comparison, a CSR was implemented in addition using an as-soon-as-possible (ASAP) schedule. The resulting numbers of operations are listed in Table I. These operations are either adder/subtractor for the realization without pipelining, or registers and registered adder/subtractor using CSR or the optimal pipelining. As expected, the fewest add/subtract resources without pipelining are obtained by H_{cub}. The adder graphs of H_{cub} AD min. result in about 6.9% more adders on average when no pipelining is applied. However, due to the lower adder depth, the least pipelined operations are achieved using H_{cub} AD min. The proposed optimal pipelining leads to a large reduction of operations in comparison to the CSR method, which are 29.1% for H_{cub} and 6.8% for H_{cub} AD min. The overhead for optimal pipelining compared to the combinational adder graph is 45.4% on average. A total number of 21 operations could be saved for H_{cub} using adder duplication but only two operations could be saved for H_{cub} AD min. The optimization time of the BILP algorithm took between 0.9 seconds and 36 minutes on an Intel Core i5 CPU with 2.4 GHz.

TABLE I
Optimization results for H_{cub} and H_{cub} AD min. using the benchmark set of [2], without any pipelining (wo. pip.), CSR pipelining and the proposed optimal pipelining (opt. pip.).

N	H_{cub} AD	H_{cub} No. of Operations wo.pip.	H_{cub} No. of Operations CSR	H_{cub} No. of Operations opt.pip.	H_{cub} AD min. AD	H_{cub} AD min. No. of Operations wo.pip.	H_{cub} AD min. No. of Operations CSR	H_{cub} AD min. No. of Operations opt.pip.
6	3	6	11	10	3	6	11	10
10	4	9	19	17	3	9	14	14
13	5	11	30	25	3	13	20	20
20	9	11	56	36	3	14	22	21
28	6	17	47	36	3	20	29	26
41	6	22	91	50	3	24	39	38
61	6	32	108	69	3	35	52	47
119	3	53	89	76	3	53	89	76
151	4	70	150	107	3	73	92	91
avg.:	5.11	25.67	66.78	47.33	3	27.44	40.89	38.11

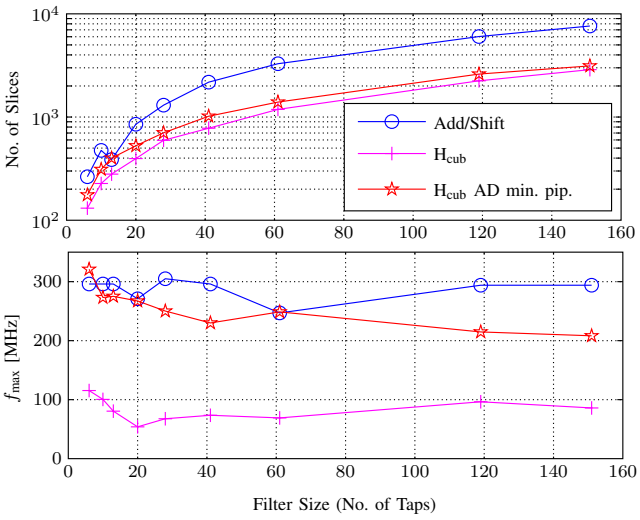

Fig. 3. Resource and speed comparison of adder graph based methods

TABLE II
SYNTHESIS RESULTS OF BENCHMARK FILTERS FROM [2] FOR THE PROPOSED METHOD (H_{CUB} AD MIN. OPT. PIP.) COMPARED TO OTHER METHODS.

		H_{cub} AD min. opt. pip.				H_{cub} wo. pip.				Add/Shift [2]		MAC				
N	w_o	FFs	LUTs	Slices	f_{max} [MHz]	FFs	LUTs	Slices	f_{max} [MHz]	Slices	f_{max} [MHz]	FFs	LUTs	Slices	DSP48	f_{max} [MHz]
6	29	344	216	176	320.92	169	215	131	115.49	264	296	58	31	36	6	402.7
10	30	531	399	311	273.15	281	394	227	100.42	475	296	59	32	44	10	402.1
13	30	701	540	400	275.10	363	507	280	80.53	387	296	59	32	41	13	406.3
20	30	941	687	527	267.31	513	661	396	53.91	851	271	72	32	57	20	405.2
28	30	1280	1044	704	250.19	772	1006	595	67.67	1303	305	72	32	60	28	401.8
41	29	1793	1470	1019	229.99	1094	1451	779	73.51	2178	296	84	31	70	41	400.6
61	30	2610	2240	1397	248.76	1670	2225	1185	69.08	3284	247	98	32	76	61	400.8
119	30	4880	4169	2609	214.82	3323	4182	2243	96.41	6025	294	394	32	370	119	372.7
151	30	5913	5462	3131	208.25	4248	5405	2880	85.92	7623	294	664	33	612	151	351.4
avg.:		2110	1803	1142	254.3	1381	1783	968	82.6	2488	288.3	173	32	152	50	394.7

B. Synthesis Results

Synthesis results were obtained for filters using H_{cub} AD min. combined with the proposed optimal pipelining. These results are compared to filters based on H_{cub} without pipelining, the Add/Shift method [2] (data taken from [1]) and transposed MAC filters generated using Xilinx Coregen FIR Compiler (v4.0). Like the Add/Shift results, all designs were synthesized for Xilinx Virtex-4 FPGAs (XC4VLX100-10 and XC4VSX55-10 for MAC-based filters) using ISE 11.1 and were optimized for speed. Inputs and outputs of the filters are registered to enable a fair speed comparison. Like in [2], the input word size has been set to 12 bit. The synthesis results are summarized in Table II. All filters were implemented with full precision, resulting in a varying output word size (w_o in Table II).

The number of slices as well as the maximum frequency for the adder graph based methods of Table II are plotted in Fig. 3. It shows that the proposed optimal pipelining based on H_{cub} AD min. outperforms the Add/Shift method in resource usage in most of the cases (except for $N = 13$) while a similar performance is achieved. On average, a reduction of 54.1% Slices is obtained while the maximum frequency is reduced by 11.8%. In relation to H_{cub}, the overhead for optimal pipelining is 18.0% slices on average, which is dominated by additional flip-flops. This overhead is small compared to the achieved average speedup of 308%.

The synthesis results for the MAC-based FIR filters are shown in the last column of Table II. The transposed structure was chosen in FIR Compiler which uses the DSP blocks for multiplication and structural adders. It can be seen that the MAC filters are beneficial for very high speed filtering as they offer a further speedup of 55.2% compared to the proposed method. However, the DSP blocks must be available, which is still a problem for low-cost FPGAs, even for modern ones.

V. CONCLUSION

It was shown in this paper that the proposed method of optimal pipelined adder graphs leads to filter designs that have much lower resource usage than comparable adder graph based methods while having a similar performance. As the Add/Shift method [1], [2] needs less resources than parallel DA implementations, one can conclude that the presented method also outperforms parallel DA implementations. Therefore, the proposed method is attractive for small FPGAs with low embedded multiplier count or high resolution filters requiring a larger word size than possible when using embedded multipliers.

The proposed method can in principle be used with any MCM algorithm. However, it was shown that a lower adder depth is much more important for resource efficient pipelining on FPGAs than a minimal adder count, which is target of most MCM algorithms. The method is not limited to FPGAs only. It may be modified for ASIC designs to consider different costs for adders and registers.

ACKNOWLEDGMENT

The authors would like to thank Shahnam Mirzaei for providing filter coefficients as well as synthesis details which helped to reproduce the results of [1] and [2]. They are also grateful to Mathias Faust from NTU Singapore for fruitful discussions and managing the benchmark filters at [10].

REFERENCES

[1] S. Mirzaei, A. Hosangadi, and R. Kastner, "FPGA implementation of high speed FIR filters using add and shift method," in *IEEE ICCD 2006*, 2006, pp. 308–313.
[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. Imran, K. Khursheed, and M. O'Nils, "On the number representation in sub-expression sharing," *IEEE ICSES 2010*, pp. 17 – 20, 2010.
[4] D. Bull and D. Horrocks, "Primitive operator digital filters," *IEE Proceedings*, vol. 138, pp. 401–411, 1991.
[5] A. Dempster and M. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 9, pp. 569–577, 1995.
[6] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplications," *ACM Trans. Algorithms*, vol. 3, no. 2, pp. 1–38, 2007.
[7] Spiral Project Website. (2011) Software/hardware generation for DSP algorithms. [Online]. Available: http://www.spiral.net
[8] U. Meyer-Baese, J. Chen, C. H. Chang, and Dempster, "A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters," in *IEEE APCCAS 2006*, 2006, pp. 1555 – 1558.
[9] K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 1999.
[10] FIRsuite. (2011) Suite of constant coefficient FIR filters. [Online]. Available: http://www.firsuite.net