Efficient Structural Adder Pipelining in Transposed Form FIR Filters

Mathias Faust*, Martin Kumm†, Chip-Hong Chang*, Peter Zipf†
* School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
† Digital Technology Group, University of Kassel, Germany
Email: {faus0001, echchang}@ntu.edu.sg and {kumm, zipf}@uni-kassel.de

***Abstract*—Pipelining is a common method to implement high speed FIR filters. While the efficient pipelining of multiplications is well understood, no attention has been paid on the pipelining of structural adders so far. The delay of structural adders becomes crucial in high speed designs as they have the largest word size in non-truncated FIR filters and typically lie in the critical path. The common pipelining method results in an excessive overhead in registers when applied to the structural adders as many additional paths have to be delayed. An efficient method for pipelining structural adders using a partially redundant number representation is proposed in this paper. With a very little area overhead of 5.4%, the throughput of the structural adders can be doubled while a speedup factor of up to 7 can be achieved with an area overhead of 26.7%.**

I. Introduction

The implementation of efficient high speed finite impulse response (FIR) filters attracted much attention. An FIR filter in transposed form can be separated into a multiplier block, structural adders (SAs) and algorithmic delays as shown in Fig. 1. The transposed form is known to be beneficial in terms of speed as the algorithmic delays cut the critical path of the SAs and, thus, act like pipeline registers. The primary focus in previous work on the optimization of FIR filters was to reduce the area of the multiplier block. In case the coefficients are constant, the multiplications can be reduced to a network of adders and binary shifts. The optimization of the multiplier block to reduce the number of adders required for its implementation is widely known as the multiple constant multiplication (MCM) problem [1]–[4]. Besides the area, the critical path delay and power consumption are important design goals. Hence, MCM methods which obtain solutions with reduced or minimal logic depth were proposed [5]–[7]. If the critical path delay cannot meet the throughput target even by using fast carry-propagate adders (CPAs) like carry look-ahead adders (CLAs), pipelining can be performed to further increase the speed. The effect of pipelining MCM blocks was studied in [8]. Heuristic and exact minimization methods to directly pipeline the MCM block were proposed [9], [10].

Carry-save arithmetic is another way to speed-up MCM operations. Different methods have been proposed for the design of carry-save adder (CSA) based FIR filters, including the use of CSAs for structural adders [11], [12]. However, more adders are required compared to multiplier blocks designed by CPAs for word lengths of coefficients larger than 9 bits [12]. Moreover, the use of the redundant carry-save representation doubles the number of algorithmic registers as both the sum and carry vectors have to be saved.

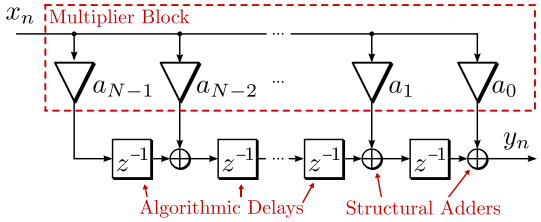

Fig. 1. Structure of an FIR filter in transposed form

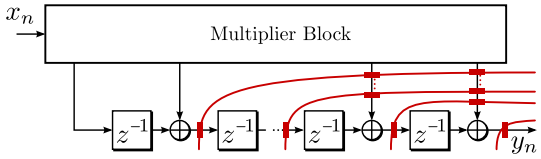

Fig. 2. Pipelining the structural adders

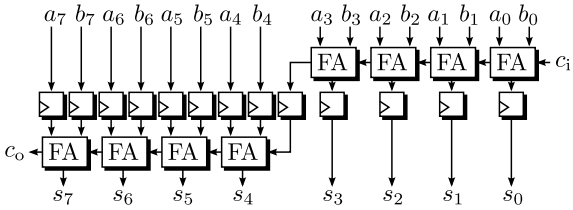

Fig. 3. Pipelined ripple-carry adder

The area optimization of SAs in transposed form FIR filters was first investigated by Faust and Chang in [13]. To the best of our knowledge, this is another pioneering work where pipelining of structural adders is considered.

II. Pipelining of Structural Adders

The conventional approach is cut-set retiming (CSR) [14]. Here, the data flow graph (DFG) is divided ('cut') into two subgraphs. If each edge from one subgraph to the other has the same direction, registers can be added to each edge without changing the functionality of the circuit (except increasing the latency). To perform CSR to the SAs in a generic way, all cuts have to be placed outside the multiplier block as illustrated by the red lines in Fig. 2. Each additional register is indicated by a small rectangle. It can be observed that just doubling the

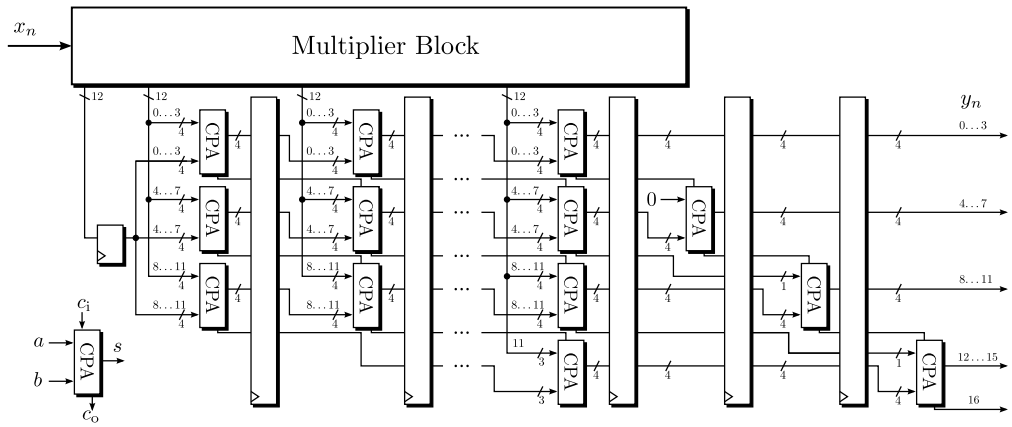

Fig. 4. Proposed structural adder pipelining architecture

registers after each SA introduces many additional registers between the multiplier block and the SAs. More precisely, an N-tap FIR filter requires $1+2+\ldots+N-2=\frac{1}{2}N^2-\frac{3}{2}N+1$ additional registers for pipeline balancing to include a single register to each SA. Of course, there may be fewer registers required when cuts through the multiplier block are allowed. However, for symmetric filters where $a_i = \pm a_{N-1-i}$, there are still many horizontal connections which have to be cut leading to additional registers for pipeline balancing. Besides that, more connections have to be cut in the horizontal direction when the multiplier block is based on a shift-and-add realization due to the sharing of intermediate results. Another drawback lies in the pipelined ripple-carry adder (RCA) itself. Fig. 3 shows an 8-bit pipelined RCA with a critical path delay of four full adders (FAs). Again, many additional flip flops (FFs) have to be used to balance the pipeline.

III. Proposed Structural Adder Pipelining

A. Architecture

This work proposes to divide the SAs into blocks of smaller CPAs in order to *save* generated carry bits, which are passed to the next stage instead of propagating to the next full adder in the same stage. The next stage receives the carries generated from the previous stage and adds them to the correct bit position by the otherwise unused carry inputs. In other words, pipelining is carried out across the SAs instead of within each adder.

The principle is illustrated in Fig. 4, where the CPAs are divided into blocks of 4-bit CPAs. The outputs from the multiplier block are assumed to be 12 bits in this example. The orientation of the input/output signals of each CPA is shown in the subfigure at the bottom left hand corner of Fig. 4, where the carry-in (c_i) is input into the CPA from the top and the carry-out (c_o) is produced at the bottom. Note that the word size of SAs naturally increases towards the output. This is indicated by an additional row of CPAs in Fig. 4 but more rows may be necessary.

The result produced by each stage is expressed in a *partially redundant representation* that was also used in higher radix Booth multipliers [15]. These redundant numbers are merged using a single pipelined CPA in the last three stages of Fig. 4 to obtain the final result in normal binary representation.

The only overhead compared to a non-pipelined implementation are the flip-flops required for storing one carry bit for each CPA and the final pipelined CPA. For the final CPA, one of its operands corresponds to the concatenation of all CPA sum vectors and the other operand consists of the carry bits padded by zeros. As a result, if the final CPA is realized as an RCA, most of the FAs can be replaced by half adders (HAs). Besides having low area overhead, the proposed structure needs only a few additional clock cycles for the final pipelined CPA. The additional latency is equal to the number of CPAs in the last stage minus one which is substantially lower than the implementation based on pipelined CPAs.

B. Area Overhead Estimation

In the following it is assumed that the CPAs are realized by simple RCAs. Pipelining changes the ranges of the RCA outputs in two's complement representation due to the sign extension. The calculation of the signal bit widths within the SA block has to account for the range expansion contributed by the weight of the saved carry. If it exceeds the minimum word length requirement, a sign extension error will occur. If the regular range expansion reaches the final output word length, there is no need to extend the word length of the SAs within the pipelined block. As no further sign extension of the data path can occur, the potentially incorrect sign bits can be corrected by subsequent calculation. This leads to a small variable overhead that is hard to be expressed in closed form. Therefore, only the deterministic overheads introduced by pipelining will be estimated.

Given the word length $n(a_i)$ of the SA corresponding to the coefficient a_i and the maximum allowable output word length s_CPA of each CPA (incl. carry-out), the number of CPA blocks is given by:

$$n_{b,i} = \left\lceil \frac{n(a_i)}{s_\mathrm{CPA}} \right\rceil \tag{1}$$

The overhead Δ incurred for pipelining a single SA is given

TABLE I
THEORETICAL RESULTS FOR PIPELINING FILTER 1 OF [16] WITH DIFFERENT MAXIMUM ALLOWABLE WORD SIZES OF PIPELINED CPAS

s_{CPA}	Blocks	SA	Pipeline Overhead				
		FA equiv.	FF	FA	HA	FA equiv.	Inc. %
2	13	5247	1815	131	12	1952.0	37.20
3	9	5247	1151	75	15	1233.5	23.51
4	7	5247	839	51	16	898.0	17.11
5	5	5247	587	38	12	631.0	12.03
6	5	5247	538	30	16	576.0	10.98
7	4	5247	421	27	16	456.0	8.69
8	4	5247	390	18	15	415.5	7.92
9	3	5247	295	23	15	325.5	6.20
10	3	5247	280	22	14	309.0	5.89
11	3	5247	251	11	13	268.5	5.12
12	3	5247	239	7	12	252.0	4.80
13	2	5247	152	8	12	166.0	3.16
14	2	5247	153	10	11	168.5	3.21
15	2	5247	151	12	10	168.0	3.20
16	2	5247	153	16	9	173.5	3.31
17	2	5247	155	23	8	182.0	3.47
18	2	5247	146	21	7	170.5	3.25
19	2	5247	141	22	6	166.0	3.16
20	2	5247	130	19	5	151.5	2.89
21	2	5247	117	15	4	134.0	2.55
22	2	5247	102	8	3	111.5	2.13
23	2	5247	90	2	2	93.0	1.77
24	2	5247	86	0	1	86.5	1.65

TABLE II
SYNTHESIS RESULTS OF THE EXAMPLE OF TABLE I FOR DIFFERENT SYNTHESIS SETTINGS

s_{CPA}	Min. Area			Min. Delay		Delay goal 1ns	
	Area [μm^2]	Inc. [%]	Delay [ns]	Area [μm^2]	Delay [ns]	Area [μm^2]	Delay [ns]
2	491056	33.66	1.22	706697	0.84	507343	1.00
3	448645	22.12	1.43	648691	0.86	465506	1.00
4	425586	15.84	1.78	640342	0.84	492716	1.00
5	406812	10.73	1.91	626507	0.88	485262	1.00
6	405422	10.35	2.33	640392	0.91	509322	1.00
7	395605	7.68	2.47	622097	0.96	513486	1.00
8	394770	7.46	2.88	601832	0.96	531505	1.00
9	388114	5.64	3.05	628031	0.97	553144	1.00
10	387210	5.40	3.35	597678	1.02	557621	1.01
11	383793	4.47	3.59	622303	1.02	580999	1.03
12	387532	5.49	3.99	622010	1.03	583953	1.06
13	380876	3.67	4.20	625786	1.04	577197	1.10
14	377756	2.82	4.17	642181	1.04	599354	1.06
15	378381	2.99	4.65	647610	1.05	582792	1.10
16	378415	3.00	4.80	620234	1.08	593862	1.10
17	379389	3.27	5.16	627419	1.11	602847	1.11
18	378561	3.04	5.39	612194	1.12	594315	1.12
19	378288	2.97	5.67	621727	1.14	594694	1.14
20	377270	2.69	5.93	633673	1.13	604567	1.14
21	376073	2.37	6.19	613671	1.17	616372	1.15
22	374609	1.97	6.47	601875	1.18	597877	1.18
23	372653	1.44	6.70	637528	1.18	619662	1.16
24	374213	1.86	7.04	642730	1.18	625629	1.17
TC	367381	–	6.94	620314	1.20	606629	1.20

by:

$$\Delta_{FF,a_i} = n_{b,i} - 1 \tag{2}$$

Additional pipelined stages are required for the final pipelined RCA to merge the carries. The overheads due to the additional FFs, HAs and FAs are given by:

$$\Delta_{FF,y} = (n_{b,0} - 1) \cdot n(a_0) + \frac{(n_{b,0} - 1)^2 + n_{b,0} - 3}{2} \tag{3}$$

$$\Delta_{HA,y} = (n_{b,0} - 2) \cdot (s_{\text{CPA}} - 1) + (n(a_0) \bmod s_{\text{CPA}}) \tag{4}$$

$$\Delta_{FA,y} = n_{b,0} - 2 \tag{5}$$

IV. RESULTS

A. Theoretical Results

To analyze the area overheads incurred by different granularity of pipelining, Filter 1 of [16] with 121 taps was analyzed for an input word length of 8 bits, a corresponding output word length of 25 bits and different maximum allowable word length of pipelined CPAs from 2 to 24 bits. In Table I, 'Blocks' refers to the total number of pipelined CPAs per SA, 'SA' and 'Pipeline Overhead' refer to the resources consumed by the non-pipelined SA and the overhead resources of the proposed pipeline architecture, respectively. The resources are measured in terms of the numbers of FAs, HAs and FFs. By assuming that the area of one FA is approximately equivalent to one FF or two HAs based on TSMC 0.18µm standard cell library [17], the area estimate in terms of FA-equivalent is given in 'FA equiv'. The column 'Inc. %' refers to the percentage area increment of the pipelined SA block with respect to the non-pipelined implementation.

The results in Table I show that for $s_{\text{CPA}} = 2$, thirteen blocks are required for pipelining. With increasing s_{CPA}, the number of blocks rapidly reduces to two, finally, when $s_{\text{CPA}} \geq 13$, which is approximately half of the output word length. While the overheads are expected to decrease with s_{CPA}, it is observed that for $14 \leq s_{\text{CPA}} \leq 19$, the overhead is slightly larger than that for $s_{\text{CPA}} = 13$. The reason is in the different sizes of the two adder blocks, which require a word length extension and introduce extra overhead. If s_{CPA} is increased further, the area overhead reduces because the taps further from the output are smaller than s_{CPA} and do not need to be pipelined. Ideally, the adders are split into equally sized blocks. For the example filter and input word length of eight, $s_{\text{CPA}} \in \{2, 3, 4, 5, 7, 9, 13\}$ are the best choices.

The additional area required for pipelining the example filter is less than 38% if the maximum word length of the pipelined CPA is limited to only two. The corresponding delay is the same as that of a full carry-save implementation. Using CSAs for the implementation of structural adders requires additional FFs for the algorithmic delays and one CPA to merge the redundant representation (assuming that the multiplier block results are in non-redundant representation). Then, the area overhead of equivalent FAs will be close to 50%. The overhead approaches 100% when the multiplier block results are also expressed in carry-save representation as two CSAs for each SA will be required then.

It can be observed from Table I that with a negligible area overhead of less than 4%, the s_{CPA} and, therefore, the delay of the SA block can be halved. With less than 9% area overhead, the delay can be quartered.

B. Synthesis Results

To corroborate the practical advantages of our method, a VHDL code generator for the proposed SA block was implemented and the designs were synthesized using Synopsys

Design Compiler (ver. 2007.12) using the TSMC 0.18µm standard cell library. The synthesis results of the SA block for the same filter as used in the previous section are shown in Table II. Three sets of synthesis results were generated. The first set 'Min. Area' was obtained by synthesizing the SA block with pure area optimization. Column 'Inc.' shows the percentage area increment of the SA block due to pipelining as compared to the non-pipelined two's complement implementation. The latter is given in the last row labeled as 'TC'. The synthesis results match well with the trend of area overheads predicted theoretically, but the overhead does not increase monotonically due to logic optimization. Fig. 5 shows the plot of the areas of the logic (adders), registers (flip-flops) and their combined area and the critical path delay versus the word length s_{CPA} of the CPA due to logic (adders) and registers (flip-flops). The plot is obtained based on synthesis results for area optimization. It clearly shows that the critical path delay reduces linearly with word length of CPA but the increase in total area with word length is slow and gradual.

The second set of results is obtained by synthesizing the SA block under the goal of delay minimization. The results are listed in the middle part of Table II. Under the aggressive delay optimization of Design Compiler, the delay of the reference design ('TC') is reduced from about 7 ns to 1.2 ns. This is close to the fastest design with minimum area goal (for $s_{\text{CPA}} = 2$). The price paid is an area increase of 68.68% (compared to 33.66% of the proposed architecture with minimum area goal).

The last set of results, 'Delay goal 1 ns', listed towards the right of Table II, refers to the synthesis results obtained with a target delay constraint of 1 ns. The designs with the smallest CPA word length of $2 \leq s_{\text{CPA}} \leq 9$ meet the timing closure and also have a relatively small area. For other CPA word lengths, their areas and delays are comparable to those obtained with the minimum delay goal synthesis. Interestingly, the 3-bit pipelining has the smallest area in this set of synthesis. The reason for this is that the overhead due to pipelining increases when the word length of the pipelined CPA reduces from three to two bits. The increase is more than the amount of logic that can be simplified by Design Compiler for the extra timing slack. In conclusion, the optimal size of pipelined CPA depends on the timing constraint. With the short VHDL code generation and verification time of less than three minutes, and the short synthesis time of less than five minutes per filter, it is possible to synthesize several different CPA word lengths to meet the target timing constraint.

V. Conclusion

The proposed method for pipelining the structural adders shows that the critical path delay can be drastically reduced with a very small overhead in area and latency. For the area minimization synthesis design goal, the throughput can be doubled with an area overhead of 5.4%, while the maximum speedup factor is 5.6 with an area increase of 33.66%. Using a delay minimization synthesis design goal with a fixed timing constraints, a speedup factor of about 7 can be achieved with an even smaller area overhead of 26.7%

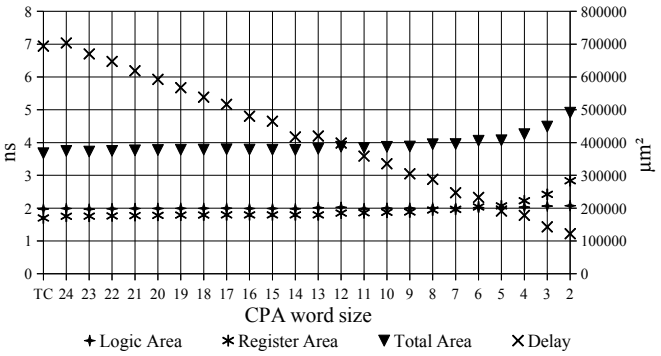

Fig. 5. Plot of the area-minimization synthesis results for different pipelining word size

References

[1] 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.

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

[3] O. Gustafsson, "A Difference Based Adder Graph Heuristic for Multiple Constant Multiplication Problems," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007, pp. 1097–1100.

[4] L. Aksoy, E. Günes, and P. Flores, "Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate," *Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151–162, 2010.

[5] A. G. Dempster, S. S. Demirsoy, and I. Kale, "Designing Multiplier Blocks with Low Logic Depth," in *2002 IEEE International Symposium on Circuits and Systems*. IEEE, 2002, pp. V–773–V–776.

[6] K. Johansson, "Low Power and Low Complexity Shift-and-Add Based Computations," Ph.D. dissertation, liu.diva-portal.org, 2008.

[7] M. Faust and C.-H. Chang, "Minimal Logic Depth Adder Tree Optimization for Multiple Constant Multiplication," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 457–460, 2010.

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

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

[10] M. Kumm, D. Fanghänel, K. Möller, P. Zipf, and U. Meyer-Baese, "FIR Filter Optimization for Video Processing on FPGAs," *EURASIP Journal on Advances in Signal Processing (Springer)*, pp. 1–18, 2013.

[11] A. Blad and O. Gustafsson, "Integer Linear Programming-Based Bit-Level Optimization for High-Speed FIR Decimation Filter Architectures," *Circuits, Systems, and Signal Processing*, vol. 29, pp. 81–101, 2009.

[12] O. Gustafsson and L. Wanhammar, "Low-Complexity and High-Speed Constant Multiplications for Digital Filters Using Carry-Save Arithmetic," in *intechopen.com*. InTech, Apr. 2011.

[13] M. Faust and C.-H. Chang, "Optimization of Structural Adders in Fixed Coefficient Transposed Direct Form FIR Filters," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2185–2188, 2009.

[14] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 1999.

[15] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic*. New York: John Wiley & Sons Inc., 2001.

[16] Y. Lim and S. Parker, "Discrete Coefficient FIR Digital Filter Design Based Upon an LMS Criteria," *IEEE Transactions on Circuits and Systems*, vol. 30, no. 10, pp. 723–739, Oct. 1983.

[17] Artisan Components Inc., *TSMC 0.18µm Process 1.8-Volt SAGE-X^{TM} Standard Cell Library Databook* , 4th ed., Sunnyvale, Sep. 2003.