Reconfigurable Multiple Constant Multiplication using Minimum Adder Depth

Mathias Faust*, Oscar Gustafsson†, and Chip-Hong Chang*
*Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore
E-mail: {faus0001, echchang}@ntu.edu.sg
†Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden
E-mail: oscarg@isy.liu.se

***Abstract*—The problem of reconfigurable multiple constant multiplication (ReMCM) is about finding an cost-effective network of shifts, additions, subtractions, and multiplexers to implement the multiplication of a single input variable with one out of several sets of coefficients. Most previous publications only focus on the problem with a single output, whereas the algorithm proposed here solves a multiple output ReMCM problem using a adder-graph based minimal logic depth approach. The use of minimal logic depth restricts the length of critical path and it was shown in previous work that minimum depth MCM is advantageous in terms of power consumption. The use of a adder-graph heuristic gives more possibilities for adder formation to reduce the total number of adders and multiplexers. For the polyphase decimation filters, the relation between filter length and decimation factor has been shown to have a influence on the implementation cost. Experimental results showed that a ReMCM can be implemented with up to 38% less area for decimation factor of 8 than a parallel implementation of the polyphase subfilters, while the single output problems can also be solved with results comparable to the known algorithms.**

I. Introduction

For the last three decades, the optimization of Multiple Constant Multiplication (MCM) problems has drawn attention from many researches [1]–[5]. In recent years, the interest in reconfigurable constant multiplication [6]–[10] has grown as the computation of all constants in parallel is not always necessary or possible. Applications like polyphase Finite Impulse Response (FIR) filters [11] or Discrete Cosine Transformation (DCT) can be modeled such that the distinct inputs are each multiplied by a different MCM block. In the case of a polyphase FIR filter with a decimation factor of four, as many MCM blocks as the decimation factor are required. Some reduction is possible if the initial FIR filter has linear phase and therefore is symmetric. Fig. 1 shows a polyphase FIR filter with a decimation of two. As long as it is allowable by the delay constraints, the filter can be operated at higher input clock rate so that the MCM block can be implemented as a reconfigurable MCM block. Generally, the length of the critical path has an impact on the switching activity. Constraining the logic depth of all individual constants to their minimum is important not only because the reconfigurable MCM block operates at the higher clock rate, but also to reduce the power consumption of a MCM block [3], [4], [12].

The transposed direct form of polyphase filters requires adders to sum the outputs of the different subfilters. If the

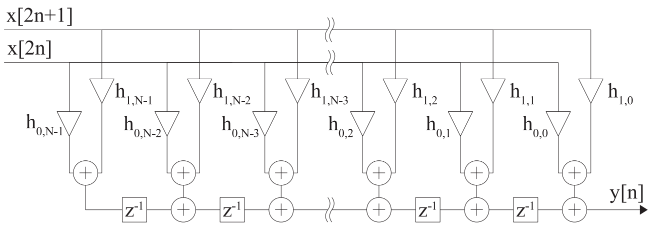


Fig. 1. Schematic for a polyphase FIR filter with a decimation factor of $M = 2$.

subfilters are merged into one single reconfigurable FIR filter which is running at the higher clock frequency, these summation adders are not required, as the summation takes place over time within the loop-back path shown in Fig. 2.

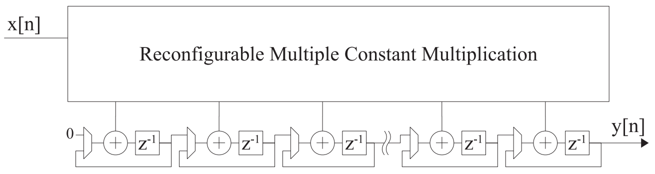


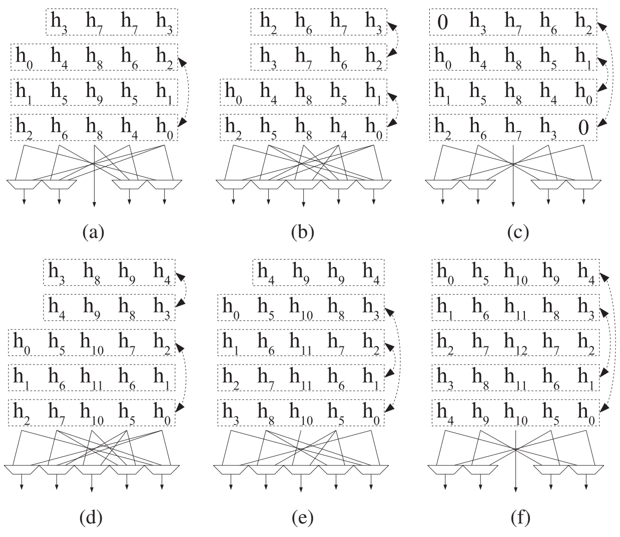
Fig. 2. Schematic for a polyphase FIR decimation filter using a reconfigurable MCM block.

For the solutions to MCM problems, two main approaches exist, which can be both adapted to Reconfigurable MCM (ReMCM). The Common Subexpression Elimination (CSE) methods, as used in [5], [10], searches for patterns among one or several representation of the constants to reduce the number of adders, while adder-graph methods, used for example in [1]–[4], [9], build a graph of adders independent of any representation. In general, the CSE algorithms produce solutions with shorter logic depth but have a higher number of logic operators. The adder-graph algorithms in [3], [4] restricts the maximal logic depth for each constant multiplier to the respective minimum and find, due to the larger search space, better solutions than CSE algorithms. Due to this, the algorithm from [4] was used as a basis for the proposed ReMCM algorithm.

This paper is organized as follows. Section II discusses the influence of the filter symmetry on the subfilters, followed by a description of the proposed algorithm in Section III. Experimental results are presented in Section IV and Section V concludes the paper.

II. Subfilter allocation

When a FIR filter has linear phase, the coefficients of the filter are symmetrical. When the filter of length N is split into the M subfilters, where M is the decimation factor, the symmetric property translates into pairwise symmetry of the subfilters. This pairwise symmetry can be, and is used to reduce the complexity of the ReMCM block, at the cost of symmetry MUX. The number of symmetric subfilters is influenced by both the filter length N as well as the decimation M. Two cases can be formed on M being even or odd. If M is even, the parity of N is relevant for the number of unique subfilters. Fig. 3(a) shows an example for N with odd parity, where the number of unique subfilters is $M/2+1$. The two non-repeated subfilters exhibit internal symmetry which simplifies the subfilters itself but the output assignment to the MUX is rather complicated. Fig. 3(b) and Fig. 3(c) show two examples for N with even parity where the number of unique subfilters is $M/2$ as all subfilters occur twice with reverse-ordered coefficients. For odd M the parity of N is irrelevant as there is always one subfilter without a symmetrical counterpart, as it is shown in Fig. 3(d), 3(e) and 3(f). In this case no optimization is possible.



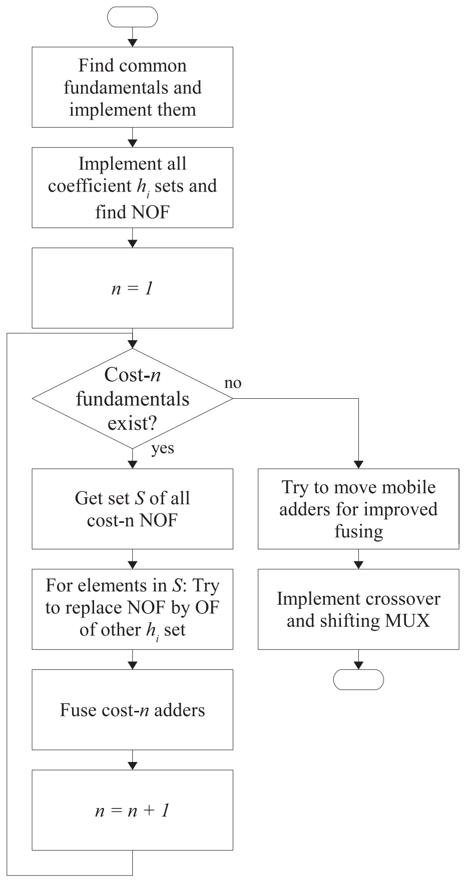
h_3 h_7 h_7 h_3
h_0 h_4 h_8 h_6 h_2
h_1 h_5 h_9 h_5 h_1
h_2 h_6 h_8 h_4 h_0
(a)
h_2 h_6 h_7 h_3
h_3 h_7 h_6 h_2
h_0 h_4 h_8 h_5 h_1
h_2 h_5 h_8 h_4 h_0
(b)
0 h_3 h_7 h_6 h_2
h_0 h_4 h_8 h_5 h_1
h_1 h_5 h_8 h_4 h_0
h_2 h_6 h_7 h_3 0
(c)
h_3 h_8 h_9 h_4
h_4 h_9 h_8 h_3
h_0 h_5 h_{10} h_7 h_2
h_1 h_6 h_{11} h_6 h_1
h_2 h_7 h_{10} h_5 h_0
(d)
h_4 h_9 h_9 h_4
h_0 h_5 h_{10} h_8 h_3
h_1 h_6 h_{11} h_7 h_2
h_2 h_7 h_{11} h_6 h_1
h_3 h_8 h_{10} h_5 h_0
(e)
h_0 h_5 h_{10} h_9 h_4
h_1 h_6 h_{11} h_8 h_3
h_2 h_7 h_{12} h_7 h_2
h_3 h_8 h_{11} h_6 h_1
h_4 h_9 h_{10} h_5 h_0
(f)

Fig. 3. Examples for polyphase filters with even parity (a)-(c) and odd parity (d)-(f) for the decimation. In example (a) the alignment is not symmetric and has two not paired subfilters. (b) has full symmetry, but requires more and more complex MUX than the zero padded version shown in (c). The difference between (d) and (e),(f) is that (d) requires more complex MUX, but could be brought to the same complexity as in (f) by padding the coefficient by a single zero.

Besides the number of unique subfilters, the number and fan-in of the MUXs required to select the reverse-ordered coefficients are influenced by the difference in the subfilter lengths, which are in turn influenced by N and M. In general, no more than $\lceil N/M \rceil$ 3-to-1 MUXs are required. For even M, if the parity of N is also chosen to be even and either $N = M{\cdot}c$, $c \in \mathbb{N}$, or zero padding on both ends is allowed, the number of MUXs is reduced to $\lfloor (N+2{\cdot}p)/M \rfloor$, where p is the number of zeros used for padding on each side, and the fan-in of the MUX is reduced from 3 to 2. This reduction translates directly into area savings, but the zero padding comes at the cost of a higher delay for the filter function. A non-optimal implementation for the case of odd M is shown in Fig. 3(d). The reduction for odd M has an intermediate step, where the subfilter without a symmetric counterpart is either one bit longer or shorter than the other subfilters. In these cases, a few 3-to-1 MUXs are required, as shown in Fig. 3(e). If all subfilters are of the same length, the number of MUXs is also reduced to $\lfloor (N+2{\cdot}p)/M \rfloor$ 2-to-1 MUXs as shown in Fig. 3(f). The filter in Fig. 3(d) can be padded by one zero on each side to reduce the MUX complexity to the same as that of 3(f).

III. Proposed Algorithm

Based on the minimal logic depth adder graph based 'minLD' [4], a new algorithm for reconfigurable MCM was developed. The main difference between the proposed algorithms and the algorithms of Tummeltshammer *et al.* [9] and Chen *et al.* [10] is that it optimizes a complete MCM with several concurrent outputs to be reconfigurable as a set, whereas the latter algorithms [9], [10] only target MCM with a single output.



Find common fundamentals and implement them
Implement all coefficient h_i sets and find NOF
$n = 1$
Cost-n fundamentals exist?
no
yes
Get set S of all cost-n NOF
For elements in S: Try to replace NOF by OF of other h_i set
Fuse cost-n adders
$n = n + 1$
Try to move mobile adders for improved fusing
Implement crossover and shifting MUX

Fig. 4. Flow of the proposed algorithm.

The filter design phase is not part of the algorithm, therefore the design of the filter length and filter length parity is left to the filter designer, but the algorithm will, when allowed, use zero padding to optimize the complexity of the output MUX as described in the previous section. The proposed algorithm solves the MCM problems for each unique subfilter as well as the MCM problem for the overall filter. Using the information from the adder graphs for the subfilter problems, the maximal amount of required adders per Logic Depth (LD) is calculated. The Non-Output Fundamentals (NOFs) with $LD = 1$ from the overall problem are analyzed to see if there is a set of NOF that is more beneficial to all subfilters than the ones found in the separate subfilters. Doing so can reduce the number of MUXs required to generate the $LD = 1$ part. Next all adders with $LD = 1$ are fused. Before effectively fusing a NOF, the algorithm tests if the NOF could be replaced by other fundamentals required by another subfilter. If this is not possible or the fundamental is an output fundamental, a fusion routine is started. This routine calculates the cost for all possible fusions. The bit width calculation form [13] is used to obtain the width of the required arithmetic operators and MUXs and then the cost measure from [9] is used to establish the cost. When all costs are known, the fusion with the smallest incremental cost is chosen and the fundamentals are fused. This is repeated until the minimal number of arithmetic operators is achieved. Optionally, the routine allows additional arithmetic operators if further fusion does not reduce the required area, but this might result in higher power consumption and this issue is currently being investigated. When all $LD = 1$ adders are implemented and fused, the next higher LD is attained. When the fusion for all levels of adder depth is completed, the algorithm will attempt to improve the reconfigurable adder graph by moving mobile adders, but only a rudimentary implementation of this concept is done at the moment. After this, the required MUX at the output is implemented. The overall flow of the algorithm is shown in Fig. 4.

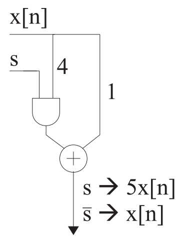


Fig. 5. Usage of an AND to bypass adder.

Further optimization ideas were tried but the implementation was not completed before the code was used to generate the results shown in the next section. Among these optimizations was that the alignment of the coefficients influences the fusion process. For polyphase filters with only two unique subfilters, the search is rather easy, as it is only a pairwise matching. For a filter with a higher number of subfilters, it is fairly complicated as for each unique subfilter, two or three choices are possible due to the symmetry MUXs shown in Figures 3 (a)-(f). Also the calculation of the savings depends often on the parts of the filter that has not yet been processed. Another idea was that if an output fundamental is located at a LD which is not the maximal LD, an output MUX might be required to select this output fundamental. This could be optimized by using a simple bitwise AND to switch between the signal input and zero of the other input of the arithmetic operator at the next LD level. This would reduce an adder to a single feed-through and a subtractor to an inverter or a feed-through, depending on the side of input used. The schematic for this operation is shown in Fig. 5.

IV. Experimental Results

To illustrate the results from the proposed algorithms we consider two different cases, both for single and multiple outputs.

A. Single Output Reconfigurable Multiplication

For comparison, the degenerated case with only one output was examined. In this case, the decimation is set equal to the number of coefficients. The first example was taken from Figure 5 of [10], where the coefficients are $\{815, 621, 831, 105\}$, and hence $M = 4$. The optimized circuit for the example from [10] is shown in Fig. 6. The proposed method reduces the bit level cost from 9192 in [10] to 7804 square micrometer (15.1%), based on the cost estimates from [9] and a input bit width of 8. When allowing one more adder at $LD = 2$, the overall cost is further reduced to 7569 square micrometer (17.6%) although the adder/subtractor is split into one adder and one subtractor. The corresponding circuit is shown in Fig. 7. This illustrates that minimal number of arithmetic operators is not always optimal, and two distinct operations might be of shorter length than the combined operation. The proposed algorithm chooses the set $\{3, 5, 7, 63\}$ instead of $\{3, 7, 9, 63\}$ for $LD = 1$. This does not result in reduction, but the different fusion of the fundamentals is slightly more advantageous as the shorter arithmetic operator is reconfigurable, while the longer one is a fixed subtractor. The use of an adder graph algorithm enables the proposed algorithm to have more options for fusion and hence a simpler fusion of arithmetic operators at $LD = 3$ is found in the given example, where only a subtractor is needed, whereas in [10] a reconfigurable adder/subtractor is required.

Two more examples were taken from Figures 7 and 8 of [6], where the coefficients are $\{362, 392, 473\}$ and $\{39, 150, 196\}$, respectively. The optimization by the proposed algorithm results in a cost of 5639 and 3473 square micrometers. For the first set, the results are better than the algorithms in [9], [10], but requires more area than [6] due to the restriction of minimal logic depth. For the second set, there is a reduction of 3.8% over [10] due to different MUX assignments and the replacement of the generation of the fused operation which generates 3 and 5.

B. Multiple Output Reconfigurable Multiplication

As an example for polyphase FIR decimation filter, a filter specification of $0.8 \cdot \pi/M$ rad and π/M rad as passband

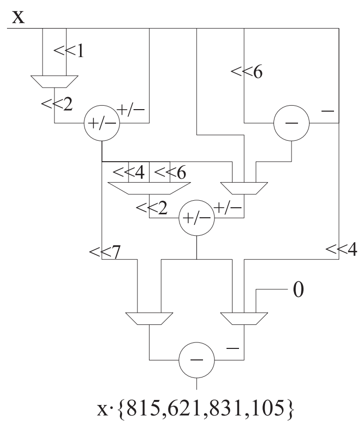


Fig. 6. Reconfigurable multiplier for the constants 815,621,831,105.

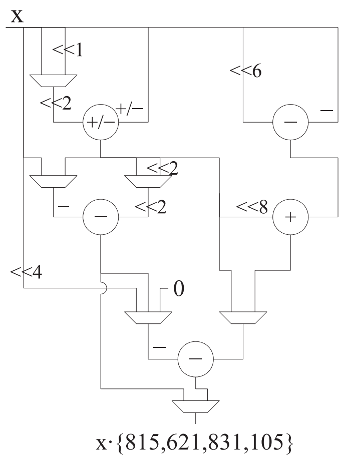


Fig. 7. Same constants as Fig. 6, but allowing one more operator. This results in two arithmetic operators at $LD = 2$ and a smaller overall area cost.

and stopband edges, respectively, and pass-/stopband ripple of 0.05 was chosen. The filter length is chosen such that it meets the specification and $N = M \cdot n$, where M is the decimation factor. After designing the filter with `firpm` with the respective length and equal weight of 1 for both pass- and stopband, the coefficients are rounded to 14 bits, which results in a maximum of 13 active bits effectively. The filter input bit width is again set to 8 bits. Table I shows the filter length for the overall filter and the subfilters, the required components as well as the bit level cost for the ReMCM block, for each unique subfilter, for a MCM block containing all coefficients and for the symmetry multiplexers. Lastly, an estimate is given for the area reduction due to the replacement of adders with a loop and MUX before the structural adders. All results for decimation of 4 to 8 are in square micrometer and based on the cost estimate from [9]. From the results in Table I, it can be seen that the fused MCM block requires between 8 and 17% less area compared with the implementation of a MCM block which contains all coefficients. With increasing decimation and filter length, the area increases moderately. The filter with $M = 8$ is twice as long as the filter with $M = 4$ but the ReMCM implementation only requires 26% more area. The savings due to reduction of adders before the structural adders increases linearly to more than three times the size of the MCM block with all coefficients. By comparing the cost of implementation of unfused subfilters with that of ReMCM, savings of up to 38% are possible. The savings at the structural adders is more than twice the size of the ReMCM block.

V. Conclusion

An algorithm which implements optimized reconfigurable multiple constant multiplication was presented. It is capable to generate a reconfigurable block that produces the product of an input multiplied by several fundamentals at any time. The method uses a minimal logic depth adder-graph algorithm internally and holds opportunities for further improvement. The experimental results show an area cost improvement of up to 38% compared with a parallel implementation of the subfilters and the results for single output ReMCM are comparable to those of the previously published algorithms.

References

[1] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *IEE Proc. G on Circuits, Devices Syst.*, vol. 138, no. 3, pp. 401–412, Jun. 1991.

[2] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, p. 11, May 2007.

[3] K. Johansson, "Low power and low complexity shift-and-add based computations," Ph.D. dissertation, Linköping University, 2008, Linköping Studies in Science and Technology. Dissertations.

[4] M. Faust and C. H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," in *Proc. IEEE Int. Symp. on Circuits Syst., 2010. ISCAS 2010.*, Paris, France, May 30 - Jun. 2 2010, pp. 457–460.

[5] F. Xu, C. H. Chang, and C. C. Jong, "Contention resolution: A new approach to versatile subexpressions sharing in multiple constant multiplications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 2, pp. 559–571, Mar. 2008.

[6] S. S. Demirsoy, A. G. Dempster, and I. Kale, "Design guidelines for reconfigurable multiplier blocks," in *Proc. IEEE Int. Symp. on Circuits Syst., 2003. ISCAS 2003.*, vol. 4, Bangkok, Thailand, 25-28 May 2003, pp. 293–296.

[7] N. Sidahao, G. A. Constantinides, and P. Y. Cheung, "Multiple restricted multiplication," *Lecture Notes in Computer Science, Field Programmable Logic and Application*, vol. 3203/2004, pp. 374–383, 2004.

[8] S. Demirsoy, I. Kale, and A. G. Dempster, "Reconfigurable multiplier blocks: Structures, algorithm and applications," *Circuits, Syst. Signal Process.*, vol. 26, no. 6, pp. 793–827, Dec. 2007.

[9] P. Tummeltshammer, J. C. Hoe, and M. Püschel, "Time-multiplexed multiple-constant multiplication," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 9, pp. 1551–1563, Sep. 2007.

[10] J. Chen and C. H. Chang, "High-level synthesis algorithm for the design of reconfigurable constant multiplier," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 12, pp. 1844–1856, Dec. 2009.

[11] O. Gustafsson and A. G. Dempster, "On the use of multiple constant multiplication in polyphase FIR filters and filter banks," in *Proc. Nordic Signal Processing Symp., 2004. NORSIG 2004.*, Espoo, Finland, 9-11 Jun. 2004, pp. 53–56.

[12] S. S. Demirsoy, A. G. Dempster, and I. Kale, "Power analysis of multiplier blocks," in *Proc. IEEE Int. Symp. on Circuits Syst., 2002. ISCAS 2002.*, vol. 1, Scottsdale, Arizona, 26-29 May 2002, pp. 297–300.

[13] K. Johansson, O. Gustafsson, and L. Wanhammar, "A detailed complexity model for multiple constant multiplication and an algorithm to minimize the complexity," in *Proc. 2005 European Conf. Circuit Theory and Design*, vol. 3, Cork, Ireland, 28 Aug.-2 Sep. 2005, pp. 465–468.

TABLE I
IMPLEMENTATION COST FOR EXAMPLE POLYPHASE FIR FILTERS WITH DIFFERENT DECIMATION FACTORS.

Decimation	4	5	6	7	8
Filter Length	44	55	66	77	88
Subfilter Length	11	11	11	11	11
Arithmetic Operators	17	17	18	16	17

Adders	3x(8 bit)
	1x(9 bit)
	2x(10 bit)
	1x(11 bit)
	1x(12 bit)
	1x(13 bit)
		2x(8 bit)
		2x(10 bit)
		1x(12 bit)
		1x(13 bit)
		2x(17 bit)
			3x(8 bit)
			3x(11 bit)
			1x(12 bit)
			1x(13 bit)
			2x(15 bit)
			1x(18 bit)
				2x(8 bit)
				1x(9 bit)
				1x(11 bit)
				2x(12 bit)
				1x(13 bit)
				1x(16 bit)
				1x(18 bit)
					3x(8 bit)
					2x(11 bit)
					1x(12 bit)
					1x(13 bit)
					2x(14 bit)
					2x(17 bit)

Subtractors	1x(11 bit)
	1x(12 bit)
	1x(15 bit)
	2x(18 bit)
	1x(19 bit)
	1x(20 bit)
		1x(11 bit)
		1x(12 bit)
		3x(16 bit)
		2x(17 bit)
		1x(19 bit)
			1x(11 bit)
			1x(17 bit)
			1x(19 bit)
				2x(16 bit)
				1x(17 bit)
				1x(20 bit)
					1x(11 bit)
					2x(17 bit)
					1x(20 bit)

Adder/Subtractors	1x(14 bit)	1x(12 bit)	1x(12 bit)
			1x(15 bit)
			2x(16 bit)
				1x(11 bit)
				1x(14 bit)
				1x(19 bit)
					1x(14 bit)
					1x(16 bit)

Multiplexers	1x(9 bit$_{2-1}$)
	3x(11 bit$_{2-1}$)
	4x(12 bit$_{2-1}$)
	6x(13 bit$_{2-1}$)
	1x(15 bit$_{2-1}$)
	2x(18 bit$_{2-1}$)
		1x(9 bit$_{2-1}$)
		1x(10 bit$_{2-1}$)
		3x(11 bit$_{2-1}$)
		3x(12 bit$_{3-1}$)
		2x(13 bit$_{3-1}$)
		1x(14 bit$_{2-1}$)
		1x(14 bit$_{3-1}$)
		1x(17 bit$_{2-1}$)
		1x(20 bit$_{3-1}$)
			1x(9 bit$_{2-1}$)
			2x(10 bit$_{2-1}$)
			3x(11 bit$_{2-1}$)
			1x(12 bit$_{2-1}$)
			1x(12 bit$_{3-1}$)
			1x(13 bit$_{2-1}$)
			1x(14 bit$_{2-1}$)
			5x(15 bit$_{2-1}$)
			1x(15 bit$_{3-1}$)
			1x(17 bit$_{3-1}$)
			1x(18 bit$_{2-1}$)
			1x(19 bit$_{3-1}$)
				2x(9 bit$_{2-1}$)
				1x(10 bit$_{2-1}$)
				3x(11 bit$_{2-1}$)
				3x(12 bit$_{2-1}$)
				1x(12 bit$_{3-1}$)
				1x(13 bit$_{2-1}$)
				1x(13 bit$_{3-1}$)
				2x(14 bit$_{2-1}$)
				1x(14 bit$_{3-1}$)
				1x(15 bit$_{2-1}$)
				2x(15 bit$_{3-1}$)
				1x(15 bit$_{4-1}$)
				1x(16 bit$_{2-1}$)
				1x(16 bit$_{3-1}$)
				1x(17 bit$_{4-1}$)
				1x(19 bit$_{4-1}$)
					2x(11 bit$_{2-1}$)
					3x(11 bit$_{3-1}$)
					2x(12 bit$_{2-1}$)
					1x(12 bit$_{4-1}$)
					1x(13 bit$_{2-1}$)
					3x(13 bit$_{3-1}$)
					1x(14 bit$_{3-1}$)
					1x(14 bit$_{4-1}$)
					1x(15 bit$_{2-1}$)
					1x(15 bit$_{3-1}$)
					1x(17 bit$_{2-1}$)
					1x(17 bit$_{3-1}$)
					1x(17 bit$_{4-1}$)
					1x(18 bit$_{4-1}$)

Symmetry Multiplexer	6x(17 bit$_{2-1}$)
	2x(18 bit$_{2-1}$)
	2x(19 bit$_{2-1}$)
		2x(16 bit$_{2-1}$)
		4x(17 bit$_{2-1}$)
		2x(18 bit$_{2-1}$)
		2x(19 bit$_{2-1}$)
			2x(16 bit$_{2-1}$)
			4x(17 bit$_{2-1}$)
			2x(18 bit$_{2-1}$)
			2x(19 bit$_{2-1}$)
				4x(16 bit$_{2-1}$)
				4x(17 bit$_{2-1}$)
				2x(19 bit$_{2-1}$)
					4x(16 bit$_{2-1}$)
					4x(17 bit$_{2-1}$)
					2x(19 bit$_{2-1}$)

Fused Subfilters	21942	23197	26095	27814	27646
Nonfused Unique Subfilters	12986 $\underline{14095}$ 27081	12383 13822 $\underline{9117}$ 35322	10785 11919 $\underline{13223}$ 35927	11949 13218 11423 $\underline{7926}$ 44516	11374 10402 12458 $\underline{10348}$ 44582
Single MCM Block	23970	28098	29326	31250	33457
Structural Adder to MUX	-41001	-57120	-73373	-89386	-105399