AUTOMATIC SYNTHESIS OF VLSI ARCHITECTURES FOR ARBITRARY LIFTING-BASED FILTER BANKS AND TRANSFORMS

Ruben Bartholomä, Thomas Greiner, Frank Kesel

Pforzheim University
Dept. Engineering and Information Technology
75175 Pforzheim, Germany
E-mail: {ruben.bartholomae, thomas.greiner, frank.kesel}@hs-pforzheim.de

Wolfgang Rosenstiel

University of Tübingen
Dept. of Computer Engineering
72076 Tübingen, Germany
E-mail: rosenstiel@informatik.uni-tuebingen.de

ABSTRACT

Recently, the conventional lifting scheme that is widely used for the construction of Wavelets and 2-channel filter banks has been extended to M-channel filter banks ($M > 2$). This extension brings up the beneficial properties of the lifting scheme to a broader range of applications, like discrete cosine transforms. There exist many hand-crafted lifting-based VLSI architectures, which mostly concentrate on a single and specific target application having a fixed data throughput and resource consumption. However, the reusability of such architectures is limited due to the lack of scalability and flexibility. To overcome this issue, we present a novel design methodology for automatic synthesis of VLSI architectures that are suitable for arbitrary lifting-based M-channel filter banks and transforms. The methodology offers wide-ranging design space exploration with varying resource consumption and data throughput tradeoffs, as it is desired in modern System-on-Chip design.

Index Terms— Design methodology, Very-large-scale integration, Lifting scheme, Wavelet transforms, Discrete cosine transforms

1. INTRODUCTION

The lifting scheme provides excellent features, namely perfect reconstruction, reduction of computational complexity and in-place computation [1]. It is widely used for the realization of the discrete wavelet transform (DWT). The classical lifting scheme deals with the factorization of a 2-channel filter bank into a product of so-called lifting steps. Recently, a generalization of the 2-channel lifting scheme to the M-channel lifting scheme ($M > 2$) has been proposed [2]. This extension brings up the features of the lifting scheme to applications that rely on an M-channel filter bank, like the discrete cosine transform (DCT), which is widely used in image and video coding applications.

Another example for a lifting-based M-channel transform is the binDCT [3]. The concept of the binDCT is the design of multiplierless approximations of DCTs with the lifting scheme. The result is an approximated DCT that provides the desired features of the lifting scheme and the DCT as well as low-cost implementations, since there are only multiplications with dyadic coefficients necessary.

Modern System-on-Chip (SoC) designs demand for parameterizable, reusable Intellectual Property (IP) cores that can be easily integrated into system specifications. However, hand-crafted architectures mostly concentrate on a single and specific target application having a fixed data throughput and resource consumption. For example, the works of [4] and [5] focus on lifting-based architectures for DWT filter banks. In [6] two architectures for calculating DCT approximations with binDCT were presented.

In this contribution, we present a novel design methodology to automatically synthesize VLSI architectures with different area and throughput tradeoffs using a single specification of a lifting-based filter bank. Hence, our methodology is applicable during design space exploration to choose the most efficient architecture for a specific application. The proposed design methodology is implemented as a high-level compilation tool that generates synthesizable VHDL code at the register transfer level (RTL).

The rest of this paper is organized as follows. In section 2 a brief introduction into the M-channel lifting factorization is given. Section 3 explains our design methodology and in section 4 we discuss some design issues of the generated hardware architectures. Section 5 presents some experimental results and in section 6 we finally conclude the paper.

2. M-CHANNEL LIFTING FACTORIZATION

Starting point for the M-channel lifting scheme is the polyphase representation of the underlying filter bank. This representation is classified by an $M \times M$ polyphase matrix as shown in the following equation:

$$\mathbf{H}(z) = \begin{pmatrix} H_{0,0}(z) & H_{0,1}(z) & \cdots & H_{0,M-1}(z) \\ H_{1,0}(z) & H_{1,1}(z) & \cdots & H_{1,M-1}(z) \\ \vdots & \vdots & \ddots & \vdots \\ H_{M-1,0}(z) & H_{M-1,1}(z) & \cdots & H_{M-1,M-1}(z) \end{pmatrix}. \quad (1)$$

The signal flow graph (SFG) for the direct implementation of the polyphase filter bank is shown in Fig. 1. The signals $X_i(z)$ and $Y_i(z)$ are the polyphase components of the input and output signals. The filters $H_{i,j}(z)$ are FIR filters with an arbitrary number of coefficients. The idea of the M-channel lifting scheme is to factorize the polyphase matrix $\mathbf{H}(z)$ into a product of special triangular matrices, which are referred to as M-channel lifting steps. An M-channel lifting step from channel j to i is defined by the following matrix operator:

$$\mathbf{\Gamma}_{i,j}[\lambda(z)] = \mathbf{I} + \lambda(z) \cdot \mathbf{e}_i \cdot \mathbf{e}_j^T, \quad (2)$$

where $\mathbf{I}$ is the $M \times M$ identity matrix, $\mathbf{e}_i$ denotes the i-th $M \times 1$ unit vector with $i, j \in \{0, 1, \cdots M-1\}$ and $i \neq j$. $\lambda(z)$ denotes the corresponding FIR lifting polynomial. The SFG for the lifting step $\mathbf{\Gamma}_{i,j}[\lambda(z)]$ is shown in Fig. 2(a). The complete SFG for a fully lifting factorized polyphase matrix can be obtained easily by cascading the

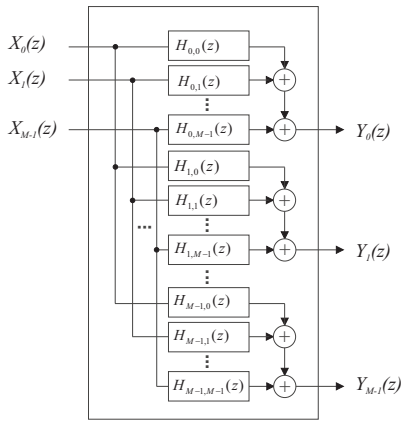

Fig. 1. SFG of a polyphase matrix $\mathbf{H}(z)$

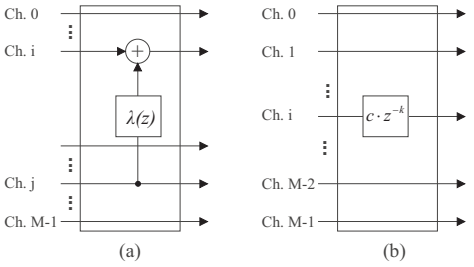

Fig. 2. (a) SFG of a lifting step $\mathbf{\Gamma}_{i,j}[\lambda(z)]$ and (b) SFG of a diagonal step $\mathbf{\Delta}_i[c \cdot z^{-k}]$

SFGs of each lifting step with appropriate values for the parameters i, j and $\lambda(z)$. In [2] it is shown that a polyphase matrix $\mathbf{H}(z)$ with $det(\mathbf{H}(z)) = z^{-k}$, $k \in \mathbb{Z}$ can be factorized into a product of lifting steps. To obtain additional flexibility during the process of factoring and designing filter banks an extension of the strict lifting factorization is proposed in [2]. This extension allows the decomposition of a matrix $\mathbf{H}(z)$ into a product of lifting steps and so-called diagonal steps. A diagonal step is defined by the matrix operator:

$$\mathbf{\Delta}_i[c \cdot z^{-k}] = \mathbf{I} + (c \cdot z^{-k} - 1) \cdot \mathbf{e}_i \cdot \mathbf{e}_i^T, \tag{3}$$

where $k \in \mathbb{Z}$, $i \in \{0, 1, \cdots M-1\}$ and c is a single dyadic coefficient of the form $\pm 2^{-n}$ with $n \in \mathbb{Z}$. The corresponding SFG can be obtained from Fig. 2(b). Equation (4) denotes the decomposition of a matrix $\mathbf{H}(z)$ into a product of N_L lifting steps and diagonal steps, respectively.

$$\begin{aligned} \mathbf{H}(z) &= \mathbf{L}_{N_L}(z) \cdot \mathbf{L}_{N_L-1}(z) \cdot \ldots \cdot \mathbf{L}_m(z) \cdot \ldots \cdot \mathbf{L}_1(z), \\ \text{with } \mathbf{L}_m(z) &= \begin{cases} \mathbf{\Gamma}_{i,j}[\lambda(z)] & \text{for lifting step} \\ \mathbf{\Delta}_i[c \cdot z^{-k}] & \text{for diagonal step} \end{cases} \end{aligned} \tag{4}$$

The factorization of (4) provides the basis for the hardware synthesis. It will be referred to as lifting specification in the following sections.

3. DESIGN METHODOLOGY

In Fig. 3 we present a block diagram of the design flow that is used for the automatic architectural synthesis. The input of this flow is

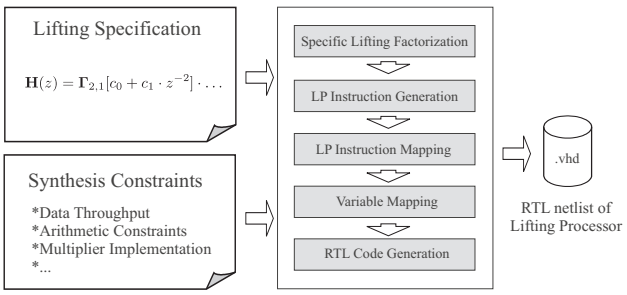

Fig. 3. Block diagram of the design flow

a lifting specification according to (4) as well as several synthesis constraints. The result is a RTL netlist of a so-called Lifting Processor (LP) architecture that calculates the operations of the filter bank, respecting to the synthesis constraints. The synthesis constraints basically enable the customization of the following aspects: desired data throughput, multiplier-based or multiplierless implementation and arithmetic precision of coefficients and data path.

The concept of our approach is to map a given lifting specification to a set of elementary operations, which allow to apply easily both parallel and pipeline techniques. The elementary operations are so-called LP instructions that are realized on an adequate LP architecture. In order to capture all the common properties of different lifting specifications we define that each LP instruction processes a lifting step or a diagonal step with exactly one polynomial coefficient. Hence, the first step in the design flow is to obtain an architecture specific lifting factorization. This factorization rewrites (4) and decomposes any lifting step $\mathbf{\Gamma}_{i,j}[\lambda(z)]$ with more than one polynomial coefficient into a product of lifting steps with only one coefficient using

$$\begin{aligned} \mathbf{\Gamma}_{i,j}[\lambda(z)] &= \prod_{\substack{k \\ c_k \neq 0}} \mathbf{\Gamma}_{i,j}[c_k \cdot z^{-k}] \text{ , with} \\ \lambda(z) &= \sum_{\substack{k \\ c_k \neq 0}} c_k \cdot z^{-k}. \end{aligned} \tag{5}$$

The result of this decomposition procedure again is a lifting specification according to (4) but the number N_L increased by the number of nonzero coefficients c_k minus one.

Depending on the precision of the polynomial coefficients, multiplierless architectures may lead to more resource efficient realizations compared to architectures that use parallel multiplier logic. Multiplierless architectures are restricted to multiplications with dyadic coefficients, which can be implemented efficiently by a single right or left shift of the multiplicands. Consequently, for the synthesis of multiplierless architectures we also have to restrict that each LP instruction processes a lifting step or a diagonal step with exactly one dyadic polynomial coefficient. Therefore, after applying (5) to the lifting specification (4) we further have to decompose each lifting step with a non-dyadic coefficient into a product of appropriate lifting steps with dyadic coefficients. Equation (6) points out this decomposition.

$$\begin{aligned} \mathbf{\Gamma}_{i,j}[c_k \cdot z^{-k}] &= \prod_{l=1}^{N_c} \mathbf{\Gamma}_{i,j}[c_{k,l} \cdot z^{-k}], \text{ with} \\ c_k \cdot z^{-k} &= \sum_{l=1}^{N_c} c_{k,l} \cdot z^{-k} \text{ and } c_{k,l} = \pm 2^{-n} \end{aligned} \tag{6}$$

Table 1. LP instruction types

Instruction Type	Arithmetic Operation	Data Shift Operation
mac	$B \leftarrow B + A * C$	NO
mul	$B \leftarrow A * C$	NO
macsft	$B \leftarrow B + A * C$	YES
mulsft	$B \leftarrow A * C$	YES

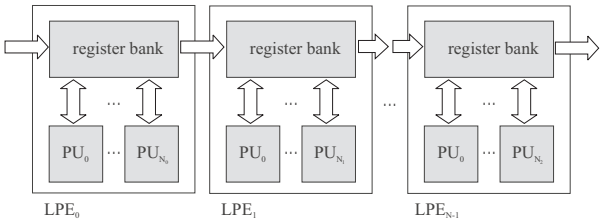

Fig. 4. Lifting Processor architecture consisting of LPEs and PUs

Again, the result of this decomposition procedure is a lifting specification according to (4), whereas the number N_L increased by $N_C - 1$.

The next step in the design flow considers the generation of an LP instruction list by iterating the decomposed lifting specification (4) in increasing order. Each element $\mathbf{L}_m(z)$ of (4) is translated into an LP instruction of a specific type. Table 1 summarizes the different types of LP instructions. If $\mathbf{L}_m(z)$ is a lifting step it is translated into an LP instruction of type *mac*. Otherwise, if $\mathbf{L}_m(z)$ is a diagonal step an LP instructions of type *mul* is generated. These two LP instruction types perform pure arithmetic operations using the operands A, B and C. The operands A and B are variables that hold information about the referred channels of the associated lifting step or diagonal step. The operand C is the only polynomial coefficient of $\mathbf{L}_m(z)$. Depending on the multiplier implementation constraint, C is either a rational or a dyadic coefficient.

During the process of LP instruction mapping the actual structure of the resulting LP architecture is determined and all the required arithmetic resources are allocated. Fig. 4 presents the structure of a synthesized LP architecture. The architecture embeds a one dimensional array of so-called Lifting Processing Elements (LPEs), where each LPE is receiving partial results from its predecessor. This array can be considered as a coarse-grained pipeline, whereas each LPE represents a pipeline stage. An LPE consists of a register bank and one or more Processing Units (PUs). Each PU has the capability to process an LP instruction within a single clock cycle. This kind of structure enables to apply both parallel and pipelining techniques for controlling the data throughput of the architecture. Parallel processing is achieved by mapping concurrent LP instructions to different PUs within a LPE and pipelining is applied by mapping LP instructions to different pipeline stages (different LPEs). With consideration of the data dependencies between the LP instructions and the synthesis constraints for data throughput, the LP instructions are scheduled and bound to the allocated PUs. We have applied a list scheduling algorithm to schedule the LP instructions of each LPE.

After scheduling and binding of the LP instructions the variables A and B of LP instructions are mapped to the data registers of an LPE's register bank. The data registers are organized as shift registers to realize the delay elements z^{-k} resulting from decomposed lifting steps or diagonal steps, respectively. To determine the clock cycle of a shift register's shift action, some of the LP instructions are

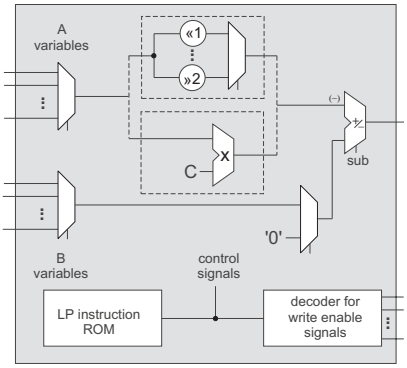

Fig. 5. Schematic diagram of a PU

modified to the types *macsft* and *mulsft*. These types are extended versions of *mac* and *mul* that initiate a shift action in parallel to the arithmetic operation. A detailed explanation is given in [7].

The final step in the design flow of Fig. 3 affects the VHDL RTL code generation for the target LP architecture.

4. PROCESSING UNIT DESIGN

Since the PU is the most important design module we want to inspect some design issues of this module. Fig. 5 outlines the schematic diagram of a PU. To process all its mapped LP instructions, each PU embeds a control unit that controls the data path elements, like multiplexers, registers or arithmetic units, as well as write enable signals that are connected to the register bank of the associated LPE. The control unit is realized as a sequencer, whereas the information of each LP instruction is stored as an instruction word in the LP instruction ROM.

The data path of a PU contains two multiplexers for selecting the data associated with the variables A and B of the corresponding LP instructions. A PU either implements a multiplier resource or an appropriate operand shifting circuit for the realization of the product $A \cdot C$, depending on the synthesis constraint for multiplier implementation. Fig. 5 outlines the two different implementation styles by the dashed boxes. In the case of a multiplier implementation a coefficient C is embedded into the instruction word of the associated LP instruction. If a multiplierless implementation is requested, the multiplications with the dyadic coefficients are realized by hardwired arithmetic shifting of the operands A. Consequently, each instruction word of an LP instruction embeds a multiplexer address to select the desired product $A \cdot C$.

5. EXPERIMENTAL RESULTS

We implemented the design flow of figure 3 as a high-level compilation tool that generates VHDL RTL code. This code can be processed by any RTL-based design flow. In order to prove the efficiency of the proposed design methodology, we present results about design space explorations of lifting-based DWT and DCT architectures, which were generated by our tool.

Equation (7) points out a lifting factorization of the popular 9/7 filter for the DWT.

$$\mathbf{H}(z) = \mathbf{\Gamma}_{0,1}[\alpha \cdot (1 + z^{-1})] \cdot \mathbf{\Gamma}_{1,0}[\beta \cdot (z + 1)] \cdot \mathbf{\Gamma}_{0,1}[\gamma \cdot (1 + z^{-1})] \cdot \mathbf{\Gamma}_{1,0}[\delta \cdot (z + 1)] \cdot diag(\zeta, 1/\zeta) \quad (7)$$

Table 2. Comparison of different 9/7 DWT architectures

Architecture	#Mult.	#Add.	#Reg.	Throughput (samples/cycles)
Proposed W1	8	8	21	2/1
Proposed W2	4	4	13	1/1
Proposed W3	2	2	9	1/2
Proposed W4	1	1	7	1/4
[5] direct	4	8	4	2/1
[5] pipelined	4	8	26	2/1

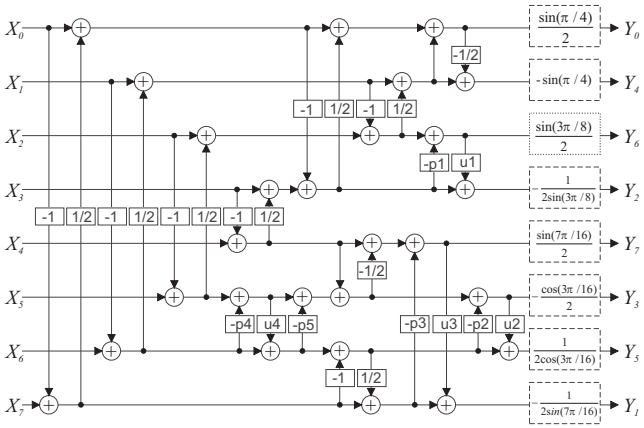

Fig. 6. Structure of a lifting factorized 8-point binDCT

The corresponding floating point coefficients can be obtained from [5]. According to [5], we also ignore the scaling coefficients in the diagonal matrix $diag(\zeta, 1/\zeta)$, resulting in a lifting specification according to (4). Using this lifting specification we generated four different multiplier-based architectures by varying the synthesis constraints for data throughput. The architectures are summarized in Table 2. They are ranging from slow low-cost architectures, with a single multiplier and adder resource, to high-speed architectures. The resource consumption is measured in terms of required multiplier, adder and register resources. The critical paths of our architectures are mainly dominated by the delay of a single multiplier and a single adder, which equals the pipelined variant of [5]. As the comparison with [5] shows, in the case of the DWT our architectures require more resources for the high-speed architectures. However, we are able to generate resource efficient low-cost architectures, which are sufficient for many applications that do not require highest speed.

As already mentioned in section 1, the concept of the binDCT is to approximate multiplierless DCTs with the lifting scheme. In Fig. 6 we present a SFG of a lifting factorized 8-point binDCT. This SFG is a slightly modified variant of the fully lifting factorized SFG from [3] that fulfills the definition according to (4). For our experiments, we defined a lifting specification using the following approximated coefficients for the lifting steps of Fig. 6:

$p1 = 3/8 \quad p2 = 5/8 \quad p3 = 1/8 \quad p4 = 3/8$
$u1 = 1/4 \quad u2 = 1/2 \quad u3 = 1/8 \quad u4 = 5/8 \quad p5 = 3/8.$

According to the proposals of [3], we ignore the scaling coefficients in the dashed boxes. To perform a design space exploration, we generated five different multiplierless architectures by simply varying the synthesis constraints for data throughput. Table 3 summarizes the results. Again, we are able to generate adequate architectures for different throughput requirements. Comparing our architecture C3 with the slow architecture of [6] shows that both architectures have the same throughput and our architecture even requires less resources. The fast architecture of [6] and our architecture C1 both have same throughput and similar resource consumptions. The critical paths of all the inspected architectures of Table 3 are mainly dominated by a single adder. Our proposed coefficients for DCT approximation reflect slightly better performance measures (smaller mean square error and higher coding gain) compared to [6].

Table 3. Comparison of different DCT architectures

Architecture	#Add.	#Reg.	Throughput (samples/cycles)
Proposed C1	17	56	4/1
Proposed C2	8	32	2/1
Proposed C3	4	16	1/1
Proposed C4	2	8	1/2
Proposed C5	1	8	1/4
[6] slow	5	40	1/1
[6] fast	20	40	4/1

6. CONCLUSION

We presented a novel design methodology for automatic architectural synthesis of arbitrary lifting-based M-channel filter banks and transforms. The methodology is suitable for design space exploration to find adequate architectures for different applications. As our experiments show, the methodology produces satisfactory results. Our generated DCT architectures are comparable to handcrafted designs.

7. REFERENCES

[1] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, pp. 245–267, 1998.

[2] Y.-J. Chen and K. Amaratunga, "M-Channel Lifting Factorization of Perfect Reconstruction Filter Banks and Reversible M-Band Wavelet Transforms," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 50, pp. 963–976, 2003.

[3] J. Liang and T.D. Tran, "Fast Multiplierless Approximations of the DCT With the Lifting Scheme," *IEEE Trans. Signal Process.*, vol. 49, pp. 141–144, 2001.

[4] B.-F. Wu and C.-F. Lin, "A High-Performance and Memory-Efficient Pipeline Architecture for the 5/3 and 9/7 Discrete Wavelet Transform," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, pp. 1615–1628, 2005.

[5] C.-T. Huang, P.-C. Tseng, and L.-G. Chen, "Efficient VLSI Architectures of Lifting-Based Discrete Wavelet Transform by Systematic Design Method," *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 5, pp. 565–568, 2002.

[6] P.P. Dang, P.M. Chau, T.Q. Nguyen, and T.D. Tran, "BinDCT and Its Efficient VLSI Architecture for Real-Time Embedded Applications," *J. of Imaging Science and Technol.*, vol. 49, pp. 124–137, 2005.

[7] R. Bartholomä, T. Greiner, F. Kesel, and W. Rosenstiel, "A Systematic Approach for Synthesizing VLSI Architectures of Lifting-Based Filter Banks and Transforms," *IEEE Trans. Circuits Syst. I, Reg. Papers*, Accepted for future publication.