Chapter 3

Parallel implementations of DSP algorithms on Field Programmable Gate Arrays

3.1 Inner product processors
3.2 FIR filters
3.3 IIR filters
3.4 Linear transformations

3.1. Parallel Synthesis of Distributed Inner Product Processors

Direct hardware implementation of large inner product operations are always difficult because of the complexity of the multiplier modules. This sub-chapter suggests a new multiplier synthesis method for this type of arithmetic operation. The fully concurrent, bit serial vector multiplication architecture is intended for FPGA or ASIC implementation. The bit level optimization algorithm exploits the inherent bit pattern coincidences of the multiplier coefficients and using the deepest descent iteration steps, generates the minimal hardware implementation of the vector multiplications. The merged, distributed arithmetic module results in more than 30% resource savings. Higher bit clock rate can be achieved due to the smaller area and reduced average fan-out. An example digital FIR filter architecture is synthesized using the algorithm and implemented on the Xilinx 4000 FPGA family.

3.1.1. The inner product operation

In many signal processing applications the calculation of the inner products of vectors are the most complicated and time/area consuming operations. Consider the inner product $\boldsymbol{y}$ of two vectors $\boldsymbol{x}$ and $\boldsymbol{h}$ given by

$$y = x^T h = \sum_{i=1}^{N} x_i h_i \qquad \textbf{(3.1.1)}$$

where $\boldsymbol{x}$ is a variable and $\boldsymbol{h}$ is an a priori known constant series. The obvious evaluation of Eq. 3.1.1 over i requires $\boldsymbol{N}$ multiplications and $\boldsymbol{N-1}$ additions. In this algorithm every product is calculated separately and then accumulated. To minimize roundoff noise effects full-precision accumulation should be applied. The modern DSP processors with internal parallel hardware multiplier/accumulator (MAC) architecture support this type of computation, what can be characterized as word serial-bit parallel computation scheme.

If $\boldsymbol{x}_i$ and $\boldsymbol{h}_i$ have $\boldsymbol{B}_x$ and $\boldsymbol{B}_h$ bit binary representations, respectively, then one obtains Eq. 3.1.2, where $\boldsymbol{x}_{i,j}$ and $\boldsymbol{h}_{i,k}$ are the j-th and k-th binary coefficient of $\boldsymbol{x}_i$ and $\boldsymbol{h}_i$.

$$y = \sum_{i=1}^{N} \sum_{j=0}^{B_x-1} \sum_{k=0}^{B_h-1} x_{i,j} h_{i,k} 2^{-(j+k)} \qquad \textbf{(3.1.2)}$$

Many alternative realizations exist. When the bits of $\boldsymbol{x}_i$ words are available simultaneously, then rewriting Eq. 3.1.2, another computation method can be derived,

$$y = \sum_{k=0}^{B_x-1} \sum_{i=1}^{N} \sum_{j=0}^{B_h-1} h_{i,j} 2^{-j} x_{i,k} 2^{-k} = \sum_{k=0}^{B_x-1} f(x_{i,k}) 2^{-k} \qquad \textbf{(3.1.3)}$$

where the function $\boldsymbol{f(x_{i,k})}$ is given by

$$f(x_{i,k}) = \sum_{i=1}^{N} h_i x_{i,k} \qquad \textbf{(3.1.4)}$$

In this formula, the different values of $\boldsymbol{f(x_{i,k})}$ can be precomputed, quantized to the wordlength $\boldsymbol{B}_x$ and stored in a 2^L-word size look-up table. During calculation the $\boldsymbol{N}$-tuple $\boldsymbol{x}_{i,k}$ is presented as an address to the memory, the precomputed value of $\boldsymbol{f(x_{i,k})}$ outputted and added to the rightshifted earlier partial sum. This type of calculation of the inner product is called distributed arithmetic (DA), as there are no distinct products of $\boldsymbol{h}_i\boldsymbol{x}_i$ anymore. The concepts was introduced in [Pel74], the theoretical background and some different structures was discussed in [Bur77]. DA is very flexible and explains the trade off between memory and arithmetic. Basically it falls into the word parallel, bit serial method, so the operation needs minimum $\boldsymbol{B}_x$ computation cycle. Depending on $\boldsymbol{N}$ and $\boldsymbol{B}_x$ the look-up table based DA arrangement can be comparable in speed with the first solution.

The purpose of this paper to introduce a more deeper analysis of inner product operation down to the bit level and at the same time to give efficient computational algorithm specifically for direct hardware implementation on FPGA or ASIC devices. Until now only memory based DA method was used, because the availability of the functional units. FPGA technology now provides

and at the same time, needs different methods to realize digital arithmetic. The basically gate-array like internal architecture of FPGAs consists of relatively simple, but powerful logic blocks or modules and programmable routing resources.

3.1.2 Bit level analysis of the inner product operation

There are other possible computations of Eq. 3.1.1. As it was shown in [Bur77], one dimensional coefficient vectors represent two-dimensional data structures, where the rows are the binary bit string representations of the coefficient words with an implicit + sign between the matrix row elements. In that sense the inner product operation is a special vector matrix product, where the elements of the $\boldsymbol{H}$ matrix are formed by the value set of $\boldsymbol{h}_{i,j}$, in two's complement notation $\boldsymbol{h}_{i,j} \in \{0,1\}$ Eq. 3.1.5.

$$[x_1 x_2 .. x_N] \begin{bmatrix} h_{1,1}2^0 & h_{1,2}2^{-1} & . & h_{1,\boldsymbol{B}_h-1}2^{-\boldsymbol{B}_h-1} \\ h_{2,0}2^0 & h_{2,1}2^{-1} & . & h_{2,\boldsymbol{B}_h-1}2^{-\boldsymbol{B}_h-1} \\ . & . & . & . \\ . & . & . & . \\ h_{N,0}2^0 & h_{N,1}2^{-1} & . & h_{N,\boldsymbol{B}_h-1}2^{-\boldsymbol{B}_h-1} \end{bmatrix} \quad \textbf{(3.1.5)}$$

An obvious implementation for the product is the application of $\boldsymbol{N}$ serial-parallel multipliers with $\boldsymbol{B_h}$ wordlength followed by the summation of the serial product outputs [Ait91]. This hardware implementation can be built up from two types of functional units: serial adders and bit delays. Non-zero elements of the $\boldsymbol{H}$ matrix show the number of partial products (data words times one bit of coefficient) in the final result and in fact gives the figure of computational complexity of the vector product. Any reduction in the $\boldsymbol{H}$ matrix elements will decrease the complexity of the computation and of course the required hardware.

The application of the Canonical Signed Digit (CSD) code [Rei60] for the a priori known h_i coefficients is the first step to decrease the complexity. Using CSD code in a B_h-bit coefficient the number of non-zero digits are statistically $B_h/3$, allowing 0, +1 and -1 weights to each digit. The hardware cost of the bit serial subtractor is the same as the adder, so signed partial product can be handled without extra hardware. This method is widely known and applied for dedicated bit serial VLSI signal processing [Den85], and an FIR filter implementation on FPGA elements was presented in [Feh92a]. In this paper a new algorithm is introduced to further reduce hardware complexity, converting the multiply and accumulate operation to a special version of DA. The algorithm based on the distributive property of multiplication :

$$\begin{aligned} (h_{i,\alpha}+h_{i,\beta})x_n &= h_{i,\alpha}x_n+h_{i,\beta}x_n \\ h_{j,\gamma}x_n+h_{k,\gamma}x_m &= h_{j,\gamma}(x_n+x_m) \end{aligned} \qquad \textbf{(3.1.6)}$$

In Eq. 3.1.6 $h_{i,\alpha}$, $h_{i,\beta}$, $h_{j,\gamma}$, $h_{k,\gamma}$ are subpolynomials of the binary representation of h_i, h_j, and h_k respectively, such that $h_i=h_{i,\alpha}+h_{i,\beta}$, $h_{j,\gamma} \subseteq h_j$, $h_{k,\gamma} \subseteq h_k$, and $h_{j,\gamma}=h_{k,\gamma}$. The first form states that multiplication of variables by coefficients are dividable to groups of partial products, while the second one states that identical partial products groups can be merged. Merged multipliers are discussed in [Die90], but only for special, matching constant values and for small N. For arbitrary values and large N an implementation can be found in [Feh92a], without minimization. Using the identities of Eq.3.1.6, the new algorithm exploits coincidences of sub-polynomials in the binary representation if h_i and realizes them separately from the remaining bits.

A simple example is given in Eq. 3.1.7, where the h_i (i=1..4) values are (115, -58, 152, 122), respectively, for B_c=8 bit.

Using the aforementioned CSD code, in this case the first two lines of the H matrix contains the same {1 0 0 -1 0 1} patterns, disregarding from the different sign and position of them. With the insertion of one shift operation in order to delay x_1 by one cycle (this means a multiplication by 2) and a serial presubtractor a composite signal ($x_5=2x_1-x_2$) is generated and multiplied by the common bit pattern, what is the new fifth row of the modified example multiplier matrix. With the introduction of x_5, the coinciding bit patterns are removed from the first two rows of H. In the example, x_2 is fully absorbed by x_5, (the second row of H is cleared by the reduction step), so x_2 can be dropped out from the signal list after the first iteration. The reduction in the total number

$$
\begin{aligned}
y=[x_1 x_2 x_3 x_4]&\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \\
=[x_1 x_2 x_3 x_4 x_5]&\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}
\end{aligned}
\qquad \textbf{(3.1.7)}
$$

of non-zero digits in $\boldsymbol{H}$ is 3. The gain of the iteration step is always smaller than the reduction, because of the cost of the preshift and the presubtract operational units. Digits of the fifth line of $\boldsymbol{H}$ are shifted right by one position to compensate the inherent one-bit-delay of the pipeline serial subtractor.

The result will be the factorization of the $\boldsymbol{H}$ matrix, as shown in Eq. 3.1.8. The $\boldsymbol{P}$ preaddition matrix is an identity matrix completed by a column vector. The new column vector of $\boldsymbol{P}$, what defines $\boldsymbol{x_5}$, has only two non-zero elements, generally one of them is an integer power of two, expressing some shift operation before partial merging, the other is +/- 1. Equation 3.1.9 shows another interpretation of the reduction step. According to that, in each iteration a new $\boldsymbol{x}^i$ signal vector, and a new $\boldsymbol{H}^i$ matrix are generated.

$$y = x^{0T} H^0 = x^{0T} \boldsymbol{P}^0 H^1 = x^{1T} H^1 \qquad \textbf{(3.1.9)}$$

The final result after the iteration process is given in the format defined by Eq. 3.1.10.
where $\boldsymbol{M}$ is the number of iteration steps. Iteration continues until the available

$$y=[x_1\ x_2\ x_3\ x_4]\cdot$$

$$\begin{bmatrix}1&0&0&0&2\\0&1&0&0&-1\\0&0&1&0&0\\0&0&0&1&0\end{bmatrix}\begin{bmatrix}0&0&0&0&0&0&0&-1\\0&0&0&0&0&0&0&0\\1&0&1&0&-1&0&0&0\\1&0&0&0&-1&0&1&0\\0&0&1&0&0&-1&0&1\end{bmatrix} \quad \textbf{(3.1.8)}$$

$$y = x^{0T}H^0 = x^{0T}\prod_{m=0}^{M} \boldsymbol{P}^m \quad H^{M+1} \quad \textbf{(3.1.10)}$$

gain of the step is higher than one. During the iteration the non-zero elements of $\boldsymbol{H}$ are continously eliminated and the arithmetic operations are executed in the preaddition phases. The convergence is guaranteed in finite number of step, because a trivial upper limit for $\boldsymbol{M}$ is $\boldsymbol{N*B_h/2}$, the maximum number of non-zero digits in $\boldsymbol{H}$, as determined by the CSD representation.

3.1.3 The complexity reduction algorithm

The goal of the algorithm is to find the minimum cost implementation of the inner product operation for a given $\boldsymbol{B_h}$-bit coefficient quantization. The cost function is measured in the number of functional units needed to realize the operation. One architecture or technology dependent factor is taken into account by the algorithm during the iteration, the $\boldsymbol{C}$ cost ratio of the shift and the arithmetic operations. In case of bit serial arithmetic this means the cost ratio of a simple D flip-flop and of a pipe-line serial adder/subtractor.

The first step of the algorithm is to check the $\boldsymbol{h}$ input vector for trivial reductions. The reason of this step is the symmetry property of the impulse response function of linear phase FIR filters or the possible underflow of some small numerical values.

Normal reduction steps are executed on the best available gain. Selection

is based on statistics of non-zero bit coincidences. The ***Sn*** statistics matrices contain information about the number of coinciding non-zero elements of rows of $\boldsymbol{H}^i$, as follows:
Sn(l,m) is the larger of coinciding identical or contradicting bit pattern of non-zero elements from the l-th and m-th rows of $\boldsymbol{H}^i$, where the m-th row is shifted right by n positions. The diagonals of the ***Sn*** matrices are cleared, because for practical $\boldsymbol{B}_h$ values self coincidences can be ignored. In case of the former example the first two statistics matrices are given in Eq. 3.1.11. The example ***H*** matrix is also shown for easy reference

$$H=\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \tag{3.1.11}$$

$$S0=\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & -2 \\ 1 & -1 & 0 & 2 \\ 1 & -2 & 2 & 0 \end{bmatrix};\ S1=\begin{bmatrix} 0 & 2 & -2 & -2 \\ -3 & 0 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix};..\ Sn$$

The parameter n (number of statistics matrices) is the function of two upper bounds and is dynamically changed during iterations according to Eq. 3.1.12, where

$$n = MAX\{\frac{\lceil 0.5\boldsymbol{B}_h \rceil - \max|S0|}{0.5+C}\ ;\ \frac{H^i{}_{SMNZ} - \max|S0|}{0.5+C}\} \tag{3.1.12}$$

$\boldsymbol{H}^i{}_{SMNZ}$ is the second maximum number of non-zero digits in rows of $\boldsymbol{H}^i$. As the iteration proceeds, the second limit becomes dominant and following $\boldsymbol{H}^i{}_{SMNZ}$, n will decrease.

With the knowledge of the full bit statistics and the actual $\boldsymbol{x}^i$ signal list, the available maximum gain can be determined. The gain is equal to the number of bit coincidences minus $\boldsymbol{C}$ times the required bit delay. The maximum gain assigns two rows of $\boldsymbol{H}^i$ to be modified. If any modified row becomes full zero, then the appropriate row is removed from $\boldsymbol{H}^i$ and the old signal is cleared from the signal list. It should be noted, that only the two reduced and the new rows and columns of $\boldsymbol{Sn}$ statistics matrices must be recalculated in each step, not the full matrices. Iteration continues until there is no more bit coincidences in the $\boldsymbol{H}^{M+1}$ matrix. The final partial product sum is simply generated from the last $\boldsymbol{H}^{M+1}$ matrix preserving the different binary weights carried by them. All iteration steps are printed out to a document file with the following syntax, while rows of the $\boldsymbol{H}$ matrix are cancelled out:

(input signal 1) FUNCTION {(input signal 2)} -> (output signal)

For the given simple example the full iteration process is shown below, where in the left column are the iteration steps, and in the rigth column are the final product summation according to the binary weights.

```
X1 DFF -> X1D1              X6 DFF -> XS2
X1D1 SUB X2 -> X5           XS2 DFF -> XS3
X3 ADD X4 -> X6             XS3 ADD X7 -> XS4
X3 ADD X5 -> X7             XS4 DFF -> XS5
X5 ADD X6 -> X8             XS5 DFF -> XS6
X1 SUB X5 -> X9             XS6 DFF -> XS7
X4 SUB X8 -> X10  XS7 ADD -> XS8
                  XS8 ADD X9 -> XSUM0
```

The two graphics in Fig. 3.1.1, show the obvious implementation using serial-parallel multipliers and the reduced flow graph after the minimization. The implementation cost of the first version is 16 arithmetic and 18 shift unit, and for the optimized one the corresponding figures are 9 and 7.

This type of description is a special kind of distributed arithmetic realization of the inner product operation. It is more distributed than the usual

look-up table based DA. In the look-up table based DA structure only the evaluation order of the triple sum was changed, but in the ROM memory table the coefficient values remained intact.

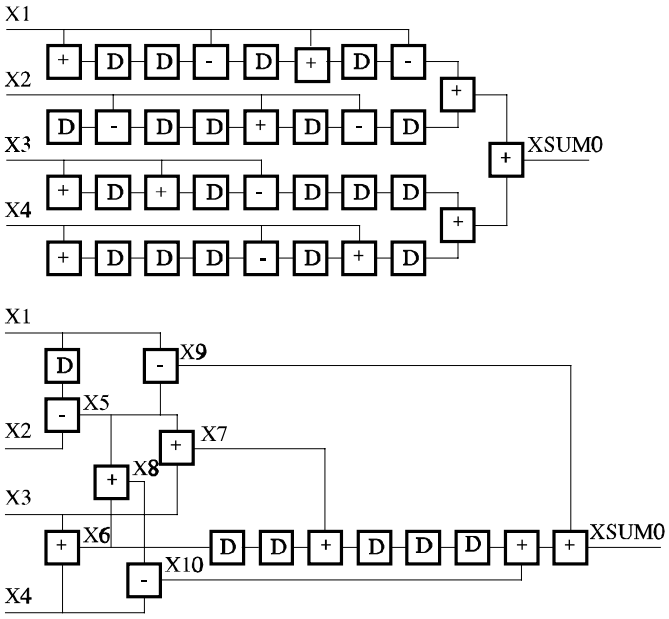

Figure 3.1.1:. The original and the generated signal flow graph

The new algorithm destroys these values down to the bit level, and builds up the signal flow graph. The arithmetic complexity is transformed in part to the signal flow graph structure. Although the necessary routing has lost its regularity, but the average fan-out and wire length are also reduced. These two factors give higher achievable clock frequency for both the FPGA and ASIC technologies. From the description of the reduced signal flow graph, any netlist format can be generated.

3.1.4. Inner product implementation on Xilinx FPGA

A typical signal processing function is selected as an example implementation. This is a linear phase low pass FIR filter, with order of N=48, and cut off frequency of Fc=.51. Data and coefficient word-length sizes are 16 bit. This filter was selected by the author in an other publication [Feh93], so the results are comparable. The selected target technology is the XILINX 4000 third generation FPGA family [Hun90]. In this device the complex Configurable Logic Blocks (CLBs) can directly realize the typical bit serial functional units, together with the carry save and the pipeline flip-flop. Further, the RAM memory configuration of the CLBs provides efficient way to built up the filter delay line of size $N*B_x$ bit. As an extension of the general algorithm the optimazation program synthesizes the full Xilinx compatible XNF netlist of the FIR filter, including the I/O interface, delay line, control signal generator and the multiplier array. The synthesis method uses the results developed in [Feh92a, Feh93 and Feh92b]. For optimal resource utilization some functions are realized by hard macros. The example FIR filter design occupies 71% CLBs of 196 of the XC4005-5 device, and the development system timing analyzer reports maximum 27 MHz bit clock frequency without any timing driven placement and routing. Most of the CLBs are used in the inner product operation. 24 CLBs are used in the preaddition, because of the FIR filter symmetry, and 65 in the generated reduced DA flow graph. This is 33% less than the complexity of the initial H matrix after the trivial reductions. Comparing the result to the other coefficient dependent look-up table based implementation [Feh92b], where the reduction is based on the magnitude distribution of the h_i values, the new method gives about 10% achievement and better operating frequency, because parallel accumulators are not needed. The layout of the FPGA chip is shown on Fig. 3.1.2.

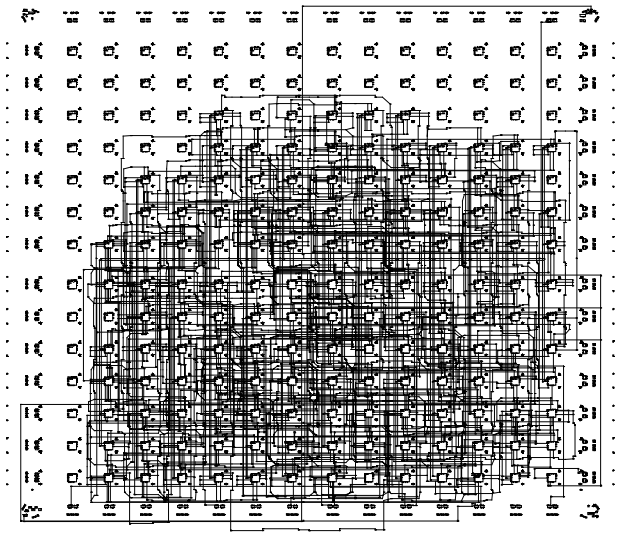

Figure 3.1.2: The layout of the example FIR filter

3.2. Parallel synthesis of FIR digital filters

A coefficient dependent multiplication scheme for Field Programmable Gate Array (FPGA) implementation of direct form Finite Impulse Response (FIR) digital filters. Using conventional design methods FPGA devices are not very efficient in digital FIR filter implementation due to the complexity of the multipliers. Analysis of filter coefficients shows that a large number of redundant leading sign bits can be find in their two's complement binary representation. Coefficient dependent multiplier logic synthesis decreases the needed amount of Configurable Logic Blocks (CLBs) by 30-40%, determined by the filter order and the selected coefficient accuracy. Memory based bit-serial distributed multiplier and accumulator units (DMACs) can be used effectively to realize multipliers with different coefficient size on the XILINX XC4000 third generation FPGA family.

3.2.1. The basic transversal FIR filter structure

FIR filters are one of the most important class of the digital filters. Although FIR filters can have recursive computation methods, practically they are always implemented as transversal filters. The signal flow graph (SFG) of the transversal filters shown in Fig. 3.2.1 is a simple tapped delay line, without any
feedback, in which the outputs of each stage, multiplied by a fixed constant,

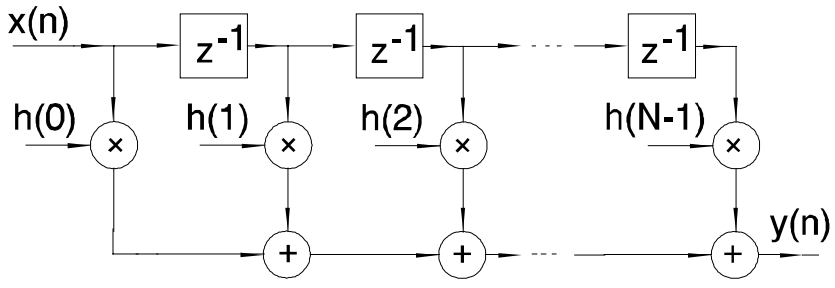

Figure 3.2.1: Transversal FIR filter

are added together. The most important advantage of these transversal filters is their inherent stability property.

The **y(n)** output sequence of a FIR filter is a linear convolution of the input data sequence **x(n)**, and the sequence of the filter coefficients **h(k)**, k=0, ... N-1.

$$y(i) = \sum_{k=0}^{N-1} h(k)\, x(i-k) \qquad i = 0, \ldots\ M+N-2; \tag{3.2.1}$$

Linear convolution is a common computational problem in digital signal processing. There are many well known fast algorithms, which reduce significantly the number of multiplications, but they are usable only in block processing mode [1].

In the obvious implementation of the convolution the computation of a given output sample needs N multiplications and N-1 additions. The multiplications are usually performed according to the data representation, but the accumulation operates on extended double precision word size to preserve numerical accuracy and avoid overflow during computation.

General purpose digital signal processors contain internal parallel hardware multiplier and accumulator (MAC) architecture optimized for this type of computation. Because of the pipe-line operation, one stage of the transversal FIR filter is executed during an instruction cycle. The same functional units are used for every computation, there are no possibility to implement application specific properties. The basic data and filter coefficient accuracy are typically equal and are between 16 and 32 bits. These processors deal with word size operands.

The internal architecture of some special signal processor, such as INMOS IMS A100 is the direct implementation of the transversal FIR filter signal flow graph. The major processing part of this chip incorporates 32 stages of the filter. The MAC units are handle 16 bits data and 4, 8, 12 or 16 bits coefficients. The multipliers internally are not fully parallel, they work with 4 bit groups of the coefficients, like the digit serial arithmetic units.

The above mentioned signal processors ignore the actual numerical values of the filter coefficients, they have general purpose hardware architecture. In the case of dedicated implementations with fixed coefficient, they numerical values can be taken into account. Analysis of the filter coefficients shows, that their dynamic range is about 2 or 3 orders of magnitude. In two's complement binary representation of small values there are a large number of redundant leading sign bits, 0s or 1s. For example, if the absolute value is smaller than 10^{-2}, the first 6 sign bits has no extra information, and a 16 x 16 bit size operation can be carried out by 10 x 16 size hardware

unit, using sign extension at the output. The next section will discuss this in depth and points out how it can be exploited to reduce the complexity of the selected memory based bit serial distributed vector multiplier accumulator (DMAC) logic [7]. The third section introduces the target FPGA device from the point of view of digital signal processing and gives a short summary of the bit-serial operational primitives, defined for the FIR filter. Detailed description is provided about the DMAC units.

To support the design procedure a direct translator program is developed. This package defines the actual block diagram of the filter according to the input parameters, determines the coefficient dependent multipliers, and synthesizes the control and I/O interface logic.

3.2.2 The coefficient dependent multiplication scheme

The same result of a multiplication can be obtained with different multiplication schemes [2], [3]. Assume the product of two 2's complement binary coded words is desired, where the data and coefficient size is B_x and B_c bits long, respectively.

$$y = hx$$

$$\textit{where} \quad h = -h_0 2^0 + \sum_{i=1}^{B_c-1} h_i 2^{-i} \quad \textit{and} \quad x = -x_0 2^0 + \sum_{j=1}^{B_x-1} x_j 2^{-j} \tag{3.2.2}$$

This gives

$$y = -h_0 x_0 + \sum_{i=1}^{B_c-1} h_i x_0 2^{-i} + \sum_{j=1}^{B_x-1} x_j h_0 2^{-j} + \sum_{i=1}^{B_c-1} \sum_{j=1}^{B_x-1} h_i x_j 2^{-(i+j)}$$

$$\textit{and} \quad y_k \in \{0 \,..\, B_x + B_c + 1\} \tag{3.2.3}$$

where **y** is a B_x+B_c-1 bit long product. (Two's complement hardware multipliers in case of fractional input operands usually generate B_x+B_c bit long product with one bit sign extension.) In the common written form, the product is given as the sum of shifted partial products. Because of the sign of $\mathbf{x_0}$, the last partial product is subtracted. For fractional input operands, the product is also true fractional and smaller than any one of the inputs. For single precision output the rounded or truncated result bits are from $\mathbf{y_o}$ to **y**.

B_{x+1}. The $\mathbf{y_1}$ product bit is already sign extension, and has the same logical value as $\mathbf{y_0}$.

In case of multiplications, where the two's complement binary code of the multiplicand is known a priori, the redundant leading sign bits (0s or 1s), except the last one, can be neglected. For positive values this statement is self evident, and for negative values, it can be seen from

$$-h_0 2^0 + h_1 2^{-1} + h_2 2^{-2} \ldots + h_L 2^{-L} = -h_L 2^{-L} \qquad \textbf{(3.2.4)}$$
$$\textit{where} \quad h_i = 1, \quad i = 0, \ldots L;$$

Assume, that $|\mathbf{h}| < \mathbf{2^{-L}}$, than the following two representation of **h** are equal.

$$-h_0 2^0 + \sum_{i=1}^{B_c - 1} h_i 2^{-i} = -h_L 2^L + \sum_{i=L+1}^{B_c - 1} h_i 2^{-i} \qquad \textbf{(3.2.5)}$$

This gives smaller partial product length and of course reduced logic complexity. Using this identity, the B_c x B_x size multiplication is feasible by (B_c-L) x B_x size hardware units. If L is comparable with B_c, then the degree of simplification is higher.

The numerical values of the elements of the impulse response sequence of the FIR filters are usually smaller than 1 and have a wide dynamic range. An example low pass filter **h(i)** coefficients are given in Table 3.2.1, the frequency response function is shown in Fig. 3.2.2, and the magnitude distribution of **h(i)**-s as a function of L is presented in Fig. 3.2.3.

Table 3.2.1. Example filter coefficients
Linear phase FIR filter, N=49, Fc=.255, Hamming window

h(0) = h(48) = 7.2633626e-004	*h(12) = h(36) = 5.2730628e-003*
h(1) = h(47) = -8.7136148e-004	*h(13) = h(35) = -1.6337261e-002*
h(2) = h(46) = -8.8238038e-004	*h(14) = h(34) = -6.4827732e-003*
h(3) = h(45) = 1.3775415e-003	*h(15) = h(33) = 2.4319301e-002*
h(4) = h(44) = 1.3249345e-003	*h(16) = h(32) = 7.6192901e-003*
h(5) = h(43) = -2.4256632e-003	*h(17) = h(31) = -3.6391525e-002*
h(6) = h(42) = -2.0347079e-003	*h(18) = h(30) = -8.6016522e-003*
h(7) = h(41) = 4.1898892e-003	*h(19) = h(29) = 5.6901874e-002*

h(8) = h(40) = 2.9711350e-003 h(20) = h(28) = 9.3591583e-003
h(9) = h(39) = -6.8818568e-003 h(21) = h(27) = -1.0193494e-001
h(10) = h(38) = -4.0750757e-003 h(22) = h(26) = -9.8369084e-003
h(11) = h(37) = 1.0785544e-002 h(23) = h(25) = 3.1690480e-001
h(24) = 5.1000648e-001

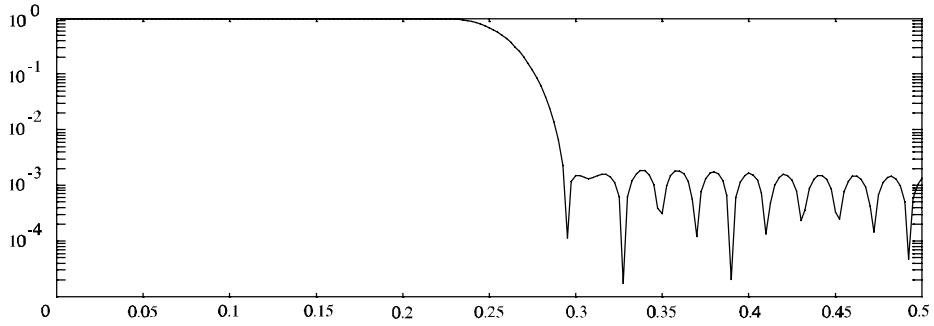

Figure 3.2.2: Frequency response of the example filter

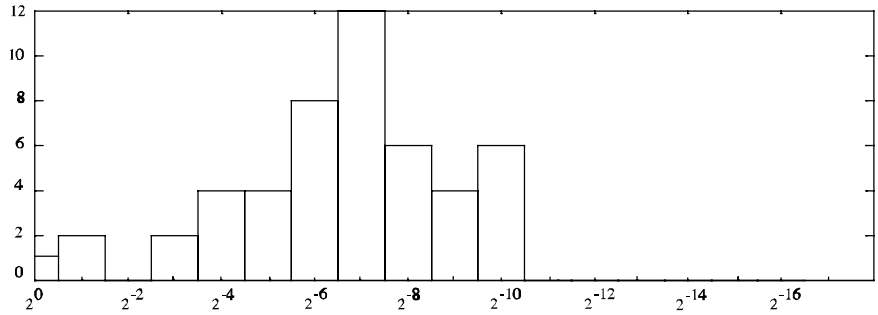

Figure 3.2.3: The magnitude distribution of *h(i)*s

Different FIR filters in type and order have been investigated and the observed distribution is similar. This directly follows from the shape of the impulse responses of the filters. It should be noted however, that as the filter order is increased or the passband width is decreased the maximum of the distribution is shifted right, raising the available reduction. According to Fig. 3.2.3, one can read out the ratio of the possible hardware reduction for the example filter. For the typical 16 bit or smaller coefficient accuracy, the

savings is significant, but in case of 20 or more bits is still more than 20%.

Fig. 3.2.3 shows that instead of 49 (or 25, if the symmetry of the linear phase filter coefficients in the given implementation can be exploited) full size multiplication, in more than half of the cases the effective bit length of the coefficients is reduced by 6 or 8 bits for this filter.

Gate complexity of state of the art FPGA logic devices [4] already approaches or exceeds the 10,000 gate level, but this is still well under the requirements of digital signal processing. Special design methods and arithmetic structures should be applied for high utilization of the available logic resources. Application of CSD codes and bit serial arithmetic also leads an area effective implementation [5], and with careful logic design high bit clock rate can be obtained.

3.2.3. The bit serial operational primitives

Basically an FPGA consists of three configurable elements: configurable logic blocks (CLBs), input/output blocks (IOBs), and programmable interconnect. The CLBs are the main resources of the logic and their complexity and flexibility has a great effect on the device performance. In the XILINX 4000 family the CLBs have two 4-input independent function generators. These function generators are implemented as memory look-up tables, allowing for the generation of any arbitrary logic function up to 4 variables. These 16x1 size memory modules in a special configuration mode can be used as RAMs also, and the read/write cycle time in that case is the same as the logic delays. In the CLBs there are two D type flip-flops with some programmable options. Both the function generators and the registers have their own inputs and outputs.

One of the most advanced feature of the 4000 family CLBs is the built-in hidden carry logic. This carry logic significantly increases the speed and efficiency of parallel adder/subtracter units. The carry logic has separate, dedicated routing resources connecting adjacent CLBs above and below each other. Using this carry logic one CLB can realize 2-bit arithmetic element.

Although the routing resources are not part of the logic, their hierarchy fits well to this filter implementation. Global networks are used for time critical control signals, long lines and short lines for normal routing.

The SFGs of the transversal FIR filters can be constructed from a limited number of operational primitives (OPs). This type of functional and hierarchical design methodology for bit-serial signal processing was discussed in [6]. The basic primitives are the bit-serial adder, subtracter,

multiplier, bit and word size delays. All these elements are treated as primitives although some of them may be very complicated logic modules. The SFG OPs library is completed with the control and I/O interface primitives. The former contains counters and special control signal generators, the later includes parallel, serial LSB first and serial MSB first I/O building blocks.

In most cases the OPs are directly mapped to the CLBs of the FPGA. The bit-serial adder and subtracter primitives need 1 CLB. The bit delays are simple D flip-flops connected as shift registers. The word size z^{-1} delays shown on Fig. 3.2.1. are realized using the CLB function generator special RAM configuration option. The hardware cost of this type of word delay is 1/2 CLB/word and 1 CLB/word up to 16 bit or 32 bit wordlength, respectively. Without this RAM option the proposed filter implementation would be practically unrealizable. The memory address generation is managed by a programmable hard macro address counter.

The most complicated OP is the multiplier. Analysis of the SFG in Fig. 3.2.1. shows that multiplications form an inner product-like structure, where the products are summed at a common junction point. This type of inner product, where multipliers are known constants, can be effectively realized by a memory look-up table based distributed vector multiplier [7]. Block diagram of the multiplier is shown in Fig. 3.2.4.

The ROM contains precomputed linear combinations of the coefficient values. The depth of the memory is determined by the number of serial input multiplicands. The width of the memory and that of the parallel adder/subtracter are equal to the multiplier coefficient effective data size. In every bit cycle a new partial product sum is added to the earlier, right-shifted value of the accumulator. If the input and coefficient data length are B_x and B_c, respectively, then the B_x+B_c-1 bit long product will be shifted out serially on SPR. For 2's complement input data representation the ADD/SUB units perform subtraction only in the B_x-th MSB bit cycle. After that the ROM output register is cleared and the most significant part of the "high precision" product scheme is shifted out with sign extension on the most significant bit position.

In the XILINX XC4000 FPGA family, because of the given 16x1 size ROM function generator the most cost effective, optimal number of the input signals for the multiplier modules are 4. Because of this (and the symmetry of the impulse response of the linear phase FIR filters) the number of the multiplier modules is $\lceil \mathbf{N/8} \rceil$. The serial product streams are summed by a binary addition tree. The hardware cost of one multiplier module is $\mathbf{B_c-L+1}$ CLBs, so it is near $\mathbf{(B_c-L)/4}$ CLBs/multiplication. Using the magnitude distribution of the impulse response to create the 4 size groups of the $\lceil \mathbf{N/8} \rceil$ multipliers, significant savings can be achieved. Different sizes of multiplier

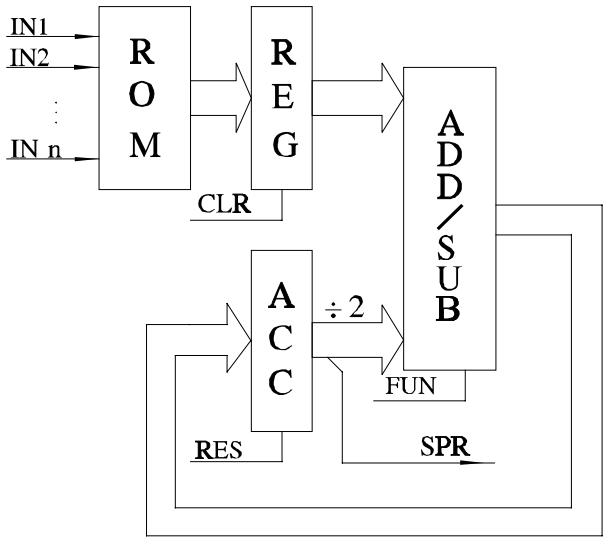

Figure3.2.4: The multiplier module

modules from 2 bit to 24 bit have been developed as library hard macro modules, because automatic logic synthesis using the dedicated carry logic is not available, only for some fixed configurations.

The filter data flow is completed with synchronizing delays and appropriate control signals. The OPs have no built in control path. An independent control network is realised using control primitives. The timing of the control signals is determined by the B_x data size and N, the filter order. The most important, time critical control signals are distributed on the whole chip using the special global networks of the FPGA.

The natural I/O format for the presented arithmetic is the bit-serial LSB first format. However, if required parallel, or bit-serial MSB first, or any combinations of these ones can be selected. These I/O options have no effect on the computational speed, only on the latency in the case of MSB first format selection.

3.2.4 Peformance analysis of synthesised FIR filter

Table 3.2.2. gives a short summary about the CLBs requirements of the example filter with and without coefficient dependent logic synthesis. In this case $\mathbf{B_x=B_c=16}$ bits were selected and the symmetry of the linear phase filter impulse response is exploited by pre addition of the equal components. Post additions generate the output signal from the serial product streams.

Table 3.2.2. CLB requirements of the example N=49 FIR filter

Hardware element	Direct method	Coefficient dep. method
Delay line	24	24
Pre additions	24	24
Multiplications	111	75
Post additions	8	8
Control	18	18
	185	149

The reduction in the multiplier logic is more than 33 %. In the second case there have left enough resources to realize any I/O option for example on the 196 CLB complexity XC4005 device. The full synchronous pipe-line architecture of the filter logic ensures high bit clock rate, according to the development system timing analyzer up to 30 MHz. The computation of a given **y(i)** output sample needs $\mathbf{B_x+B_c+\log_2(N/2)}$ bit clock period. This means about 500 kHz sampling rate for the example filter.

This subchapter has proposed a new method for digital FIR filter implementation on XILINX 4000 FPGA devices. Exploiting the redundancy in the binary representation of the filter coefficients essential complexity reduction is achieved. The flexible bit-serial DMAC arithmetic units and the programmability of the logic elements allows coefficient dependent, application specific logic synthesis. Using the suggested method a relatively high order transversal FIR filter can be realized by the XILINX XC4005 10,000 gate level device. The example filter needs about 170 CLBs. FPGA FIR filter implementations are comparable in performance by the usual DSP processor based realizations. An automatic filter design tool is developed under the MATLAB environment, which directly generates the netlist file from the filter coefficients using the bit serial operational primitive library.

3.3. Parallel synthesis of IIR digital filters

A resonator based digital filter (RBDF) implementation is presented using field programmable gate array (FPGA) elements. The realization is based on the common structure developed for recursive discrete transforms. The globally parallel property of the recursive structure is maintained, while the arithmetic operations are realized by bit-serial logic. LSB first arithmetic combined with memory table based distributed vector multipliers gives an area effective realization. Third generation FPGA devices provide efficient tools to build up DSP signal flow graphs (SFGs). An automatic filter design tool is under development, which will directly compile the netlist file from the filter coefficients.

3.3.1 The resonator based IIR digital filters

RBDFs are descended from a common structure designed to realize recursive discrete transforms [Pec86]. The structure decomposes the input signal to orthogonal components as defined by the selected recursive transformation. The parameters of the structure are the values of the basis and reciprocal basis vectors of the transformation [Pec86]. This general arrangement leads to time variant computational parameters. A typical example is the recursive Walsh-Hadamard transformation (RWHT). In the RWHT, the time variant parameters are the +1 and -1 values, so they have little effect on practical realization. The application of the RWHT to digital filtering and an implementation on FPGA elements are presented in [Pec90] and [Feh89], respectively.

The parameters of the common structure will be time invariant in the case of the recursive Fourier transform, or the RBDFs based on it. The corresponding modified structure is shown in Fig. 3.3.1. This special arrangement consists of a regular parallel connection of digital resonators supplemented by a common feedback loop. All of the resonators get the same update signal and operate independently of each other.

The practical implementation requires first order sections for real resonators, and, second order sections for complex conjugate resonator pairs. In order to reach the above mentioned good filter parameters the resonator SFGs should be selected carefully. If the state variables in the resonator sections are orthogonal to each other and magnitude-truncation quantization strategy is applied, then the filter passivity is preserved [Pec89].

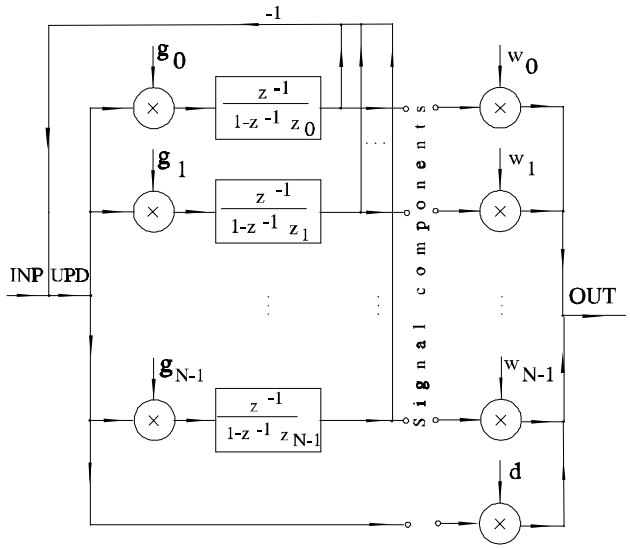

Figure 3.3.4: The common structure

A possible second order resonator block is shown in Fig. 3.3.2. This arrangement fulfils the requirements and its SFG is uniquely suitable for the implementation on the selected FPGA device using bit-serial arithmetic.
This paper presents an implementation of the common structure using third generation XILINX XC4000 FPGA elements. The 2's-complement fixed point LSB first bit-serial arithmetic fully utilizes the architectural benefits of the selected device. The the most important features of the XILINX FPGA related to DSP type design implementations can be found in Chapter 3.2, so it is not repeted here. The same is applies to the elements of the bit-serial operational primitive (OP) library defined to realize digital signal processing SFGs (see at Chapter 3.2.3). At the end of the Chapter, a short summary presents the implementation of an example 8th order elliptic bandpass filter.

To support the RBDF design procedure a direct translator program is under development. This package will define the actual block diagram of the filter according to the input parameters, determine the contents and scale factors for memory based vector multipliers and synthesize the control logic. This filter implementation tool shortens the design time to an hour.

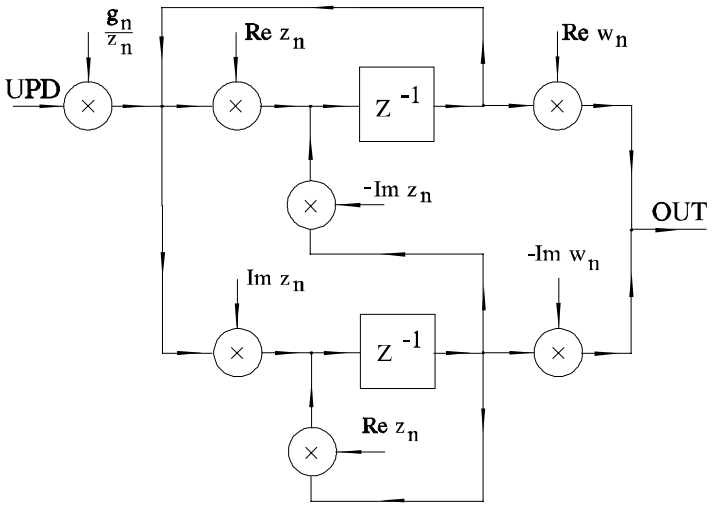

Figure 3.3.5: The second order resonator section

3.3.2 The prototype elliptic 8th order filter implementation

A prototype 8th order bandpass filter implementing the structure of Fig. 3.3.1. and Fig. 3.3.2. has been designed using conventional schematic entry tools. The selected device is the XC4005PG156, with its approximately 5000 gate density and 14 x 14 size CLB matrix. The data formats are 16 bit for the input and 12 bit for the coefficients. The four second order resonators are implemented with 8 multiplier module.The parallel adder/subtracter and accumulator unit in the multiplier was designed by the XACT Design Editor and defined as hard macro module. This design method was selected because it provides full control of the hardware resources of the CLBs. Hard macros preserve their internal partitioning and relative placement information and they are the best way to design timing-critical functional modules. The hard macro CLBs are placed in one column of the array, following the predetermined carry signal routing. Three different CLB configuration are used. The LSB and MSB logic differ from the intermediate 2-bit configuration. The LSB CLB needs the incoming carry control. The MSB CLB on the other hand is a special 2-bit block without

carry output. The netlist of the first multiplier module was generated by the design system from the schematic, and the later ones were copied and tailored from the netlist file using text editor. For all definable logic elements in these units direct placement constraints were given. ROM modules were placed directly onto the hard macro. This placement information and the regularity of the structure itself makes it possible for the XILINX Partition Placement and Route (PPR) program to finish the generation of the LCA configuration file in less than 30 minutes. The layout of the prototype filter is shown in Fig. 3.3.3.

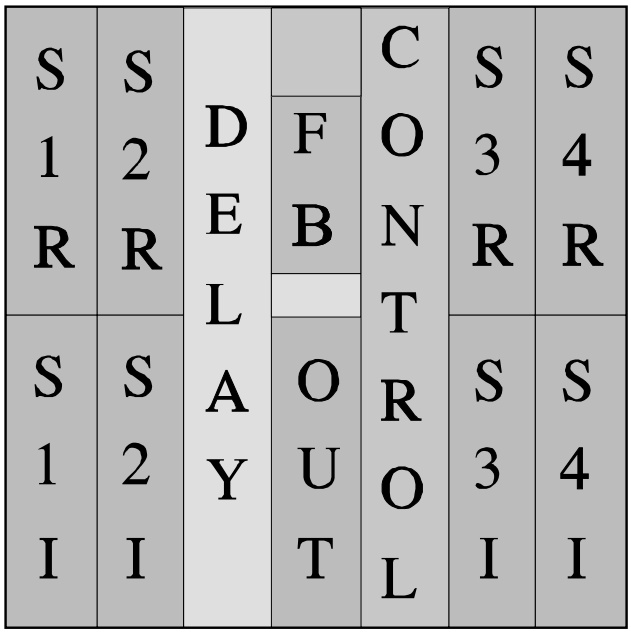

Figure 3.3.6: The layout of the prototype 8th order elliptic filter

The SxR and SxI blocks are the real and imaginary parts of the second order resonator sections, respectively. The synchronization bit delays, the truncation logic and z^{-1} word delays are grouped together (DELAY) in the centre of the chip. The feedback serial adders (FB) and the output multiplier (OUT) are also placed near the resonator outputs. The prototype filter uses the simple LSB first I/O data format. The most important control signals (CLK, RES, CLR) are distributed on primary global nets. Memory address lines in one column use the vertical long lines.

3.3.3 The automatic filter sythesiser and netlist generator

The prototype filter implementation proved that using bit-serial arithmetic, a RBDF SFG can be realized on the target device. The regularity and modularity of the RBDF suggests an algorithmic block diagram and netlist generation. All filters of this type have the same generic architecture. According to the filter order N, there are N/2 parallel second-order resonators. Occasionally first-order resonators also can occur, but only in pairs. These two first order sections are handled in a similar way, as the real and imaginary halves of a second-order section. The filter-specific blocks are the feedback, control and I/O. They are synthesized in compliance with data format. The DSP library contains only a limited number of operational, interface and control primitives. The standard, common input for the XILINX development system is the Xilinx Netlist File (XNF) format [Xil91]. The development system supports the hierarchical design method, so functional blocks can be built up from the basic primitives. These functional blocks are treated as macro symbols and only their I/O connections should be given. All information related to the design is contained in XNF files and hard macro files, which have precompiled formats. In the case of the XC4000 family, the PPR program reads the main XNF netlist file and flattens the design according to its hierarchy.

The implementation process is shown in Fig. 3.3.4.

Any filter design tool can be used to determine the filter H(z) transfer function. Fig. 3.3.4. shows MATLAB, as an example only, the development sofware needs only the parameters of the filter transfer function to be realized. The next step is to derive the parameters of the resonators from the filter H(z) transfer function following the algorithm given in [Pec89]. The filter netlist generator reads these resonator parameters from a file, or in interactive mode, from the keyboard. It needs the B_c coefficient data to calculate the contents of the multipliers look-up tables. B_c data format is also used as a selector among the different sizes of adders/subtracters. There is no possibility of building up this modules from basic CLB primitives. The B_x input data wordlength determines the timing of the control signals, and is used with the I/O mode specification to generate the selected I/O module. On demand, special interface logic can be designed and merged to the filter if the FPGA resources allows it. The netlist generator checks the program input, selects a device from the XC4000 family, and generates the XNF files. From these, the PPR will generate the FPGA configuration LCA file.

The filter netlist generator program have been written in C, the

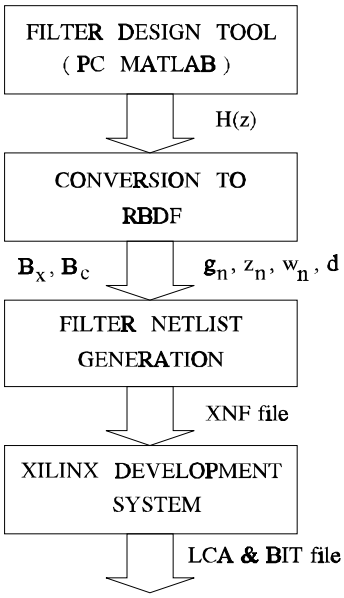

Figure 3.3.7: The automatic netlist generator program

implementation algorithm, the decomposition of the common resonator structure to second order sections and feedback summation modules are directly built in. The program generates the Xilinx technology specific netlist database and prints out all of the design parameter in a report file.

3.4. The two-dimensional DCT

The two-dimensional discrete cosine transform (2-D DCT) is one of the most effective method in image data compression. In this sub-chapter an inner product algorithm for the 8x8 2-D DCT implementation is presented. The proposed direct 2-D inner product algorithm exploits redundancies down to the bit level, and results in minimal hardware complexity. The basic algorithm separates the computation to 8 subtransform, according to the different cosine function values. Utilizing the odd-even property of the DCT, every transformed coefficients are expressed as a 4 point size inner product operation. The inner product processors are realized by an efficient distributed vector multiplication arrangement. All of the numerical parameters are built in into the inner product processors, so the arithmetic complexity is partly transformed to the internal topology of the units. The selected globally parallel, locally serial implementation style is featured by basic serial processing elements and low communication cost. It is ideal for FPGA implementation, where the available chip area is a priori partitioned between logical and routing resources. The fully concurrent bit-serial pipeline architecture needs less than 1000 arithmetic primitives. Assuming 30MHz bitclock rate in the Xilinx FPGA, the available throughput is 1 million 2-D 8x8 DCT transform/sec.

3.4.1 The DCT in image coding

Transform coding has found many applications in digital signal processing, mainly in image compression. Among the possible different transforms the discrete cosine transform (DCT) is the most widely used, due to its good energy compaction property. For highly correlated signals, especially for the first order Markov process the DCT approaches the statistically optimal Karhunen-Loeve transform (KLT). The importance of the two-dimensional DCT is emphasized by the new standards also.

Since the introduction of the transform [Ahm74], many algorithms and VLSI architectures have been proposed to improve the computation speed and reduce the hardware complexity [Nar78], [Lee84], [Kam82]. These fast algorithms can be classified into four categories, according to [Liu93], as: matrix factorisation, recursive computation, indirect computation

and systolic structure implementation. The matrix factorisation algorithms decompose the DCT into multiplications of many sparse matrices, so the necessary arithmetic operations are dramatically reduced. The recursive computations use the lower order DCT output results to calculate the higher order ones. It should make clear that the recursive property of this algorithms are not refers to time recursive operation, it simply express the calculation method. Indirect algorithms are usually apply the existing fast algorithms of the discrete Fourier transform (DFT) or the Walsh-Hadamard transform. Systolic structure implementations are similar to systolic architectures of other transforms also. These arrangements are usually better for VLSI implementation than the above mentioned recursive algorithms, because they need no global communications.

As the DCT is orthogonal, the 2-D transform is usually accomplished by two separated 1-D DCT, first along the rows, after along the columns. This conventional method requires 2N 1-D DCT's for the computation of the 2-D DCT. For parallel hardware implementation of this method, a complicated matrix transposition architecture as well as 2N 1-D DCT modules are required. Thus for more efficient computation or parallel implementation of the 2-D DCT, the algorithms that work directly on the 2-D data set have been introduced [Kam82],[Duh90],[Cho91]. These algorithms are based on matrix factorisation or indirect computation using the polynomial transform FFT, and all work in the usual block processing mode. Although they reduce significantly the number of multiplications relative to the conventional method, their arithmetic complexity prohibits the direct hardware implementation.

The new algorithm proposed in this paper is based on the inner product calculation of the 2-D transform coefficients. To hold the arithmetic complexity of the parallel architecture at a reasonable level, serial arithmetic is used to perform the computation. A fully pipe-lined structure based on an efficient method for inner product calculations [Feh93] allows high throughput capability. The full hardware structure is built up from basic arithmetic primitives and doesn't make use of multipliers. The algorithm works in the usual block processing mode. The computation needs all of the input data samples and generates the output results at the same time. This can be classified as a parallel input, parallel output (PIPO) algorithm.

3.4.2 The inner product algorithm of the 2D DCT

The NxN size 2-D DCT is defined as

$$[X] = [C]\ [x]\ [C]^T \tag{3.4.18}$$

where [x], [X] and [C] are the input, output and transform matrices with size of NxN. Definition of C is given in [Ahm74]. Most of the implementations of the 2-D DCTs are based on this transposition method [Liu93]. These architectures in case of parallel hardware implementation needs 2N 1-D DCT modules and a complicated matrix transposition unit.

Substituting the transform matrix elements, the immediate computation of [X] output data matrix from the [x] input is given by Eq. 3.4.2.,

$$X(u,v) = \frac{2}{N} c(u)\, c(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i,j) \cos\frac{(2i+1)u\pi}{2N} \cos\frac{(2j+1)v\pi}{2N} \tag{3.4.19}$$

$$u,v,i,j = 0,1,...,N-1; \qquad c(u),c(v) = \begin{cases} \sqrt{(2)} & u,v=0 \\ 1 & u,v\neq 0 \end{cases}$$

where x(i,j) and X(u,v) represent the (NxN) arrays in the data and DCT domain. Eq. 3.4.2. also can be written in matrix form.

$$X_L = [F]\ x_L \tag{3.4.20}$$

where X_L and x_L are linear sequences rearranged in lexicographic order of 2-D data and transformed array x_L={x(0,0), x(0,1), ... x(0,N-1), x(1,0), ..., x(N-1,N-1)}, X_L={X(0,0), X(0,1), ... X(0,N-1), X(1,0), ..., X(N-1,N-1)}. The [F] (N^2,N^2) matrix is the forward 2-D transform matrix, its elements are determined by Eq. 3.4.2. as shown in Eq. 3.4.4.

$$F(m,n) = \frac{2}{N} c(Q(m,N))c(R(m,N)) \cos\frac{(2Q(n,N)+1)Q(m,N)\pi}{2N} \cos\frac{(2R(n,N)+1)}{2N} \tag{3.4.21}$$

where m=0,1,...,N^2-1, n=0,1,...,N^2-1, and Q(n,N), Q(m,N), R(n,N) and R(m,N) are the quotient and remainder of the integer division operation of m/N, n/N, and stand for the original i,u row and j,v column indices, respectively. According to Eq. 3.4.4. the components of X_L can be calculated

as an NxN size inner product operation. Analysis of the 2-D transform matrix shows high degree of symmetry and redundancy between the matrix elements. Application of the basic trigonometric identity of Eq. 3.4.5. leads more efficient computation of the inner products.

$$\cos a \cdot \cos b = \frac{1}{2}\{\cos(a+b) + \cos(a-b)\} \qquad \textbf{(3.4.22)}$$

The elements of the 2-D transformation matrix [F] can be written as the sum of two term using Eq. 3.4.5. The same conversion can be applied to the c(Q(m,N))c(R(m,N)) multiplier constant values also. When Q(m,N) and/or R(m,N) are equal to zero, then the c constants take the value of $\sqrt{2}/2$. This value is equal with cos((N/2)π/2N), so these elements of [F] has the same form as the others.

For the F(m,n) matrix elements in a given row, the Q(m,N) and R(m,N) are constant values, even or odd numbers. so the multiplier factors of π/2N can have only even or odd values in the inner product. Exploiting the symmetry and periodicity of the cosine function the N^2 size inner products have only N/2 (or less) numerically different multiplier factors.

The transformation matrix [F] is separable to the sum and difference matrices, according to Eq. 3.4.5.

$$X_L = ([F^+] + [F^-])\,x_L = [F^+]\,x_L + [F^-]\,x_L \qquad \textbf{(3.4.23)}$$

These separation can be continued until all the matrices contains only one Cx numerical constant value.

$$X_L = ([F0^+] + [F1^+] + \ldots + [F(N-1)^+])\,x_L + ([F0^-] + [F1^-] + \ldots + [F(N-1)^-])\,x_L \qquad \textbf{(3.4.24)}$$

In the $[Fx^+]$ and $[Fx^-]$ matrices, for a given m,n index pair, m,n=0,1,..N^2-1, there is maximum one nonzero element, so the matrices can be merged.

$$X_L = ([F0] + [F1] + \ldots + [F(N-1)])\;x_L = \sum_{k=0}^{N-1} [Fk]\;x_L \qquad \textbf{(3.4.25)}$$

Eq. 3.4.8. already defines the algorithm, but a more better form available, using the following notation for the multiplier factors.

$$[Fx] = Cx \cdot [Tx], \quad where\ Cx = \cos\frac{x\pi}{2N}\ ;\ x = 0,1,..N-1 \tag{3.4.26}$$

Removal of the Cx multipliers from the [Fx] gives the [Tx] matrices, x=0,1,..N-1. These matrices have only +1, -1, and 0 elements. The computation of the 2-D DCT consist of two phases, as shown in Eq. 3.4.10. and Fig. 3.4.1. At first, in the pre-addition phase, the lexicographical input data is arranged according to the pattern of the nonzero elements of the [Tx] matrices. The pre-additions need only simple butterfly stages on the x_L input data vector components, and provides the necessary data combinations for the final, inner product processors.

$$X_L^{EVEN} = \sum_{k=0}^{N/2-1} C(2k)\,SCALESYM150\{[T(2k)]\ x_L SCALESYM150\} \tag{3.4.27}$$

$$X_L^{ODD} = \sum_{k=0}^{N/2-1} C(2k+1)\,SCALESYM150\{[T(2k+1)]\ x_L SCALESYM150\}$$

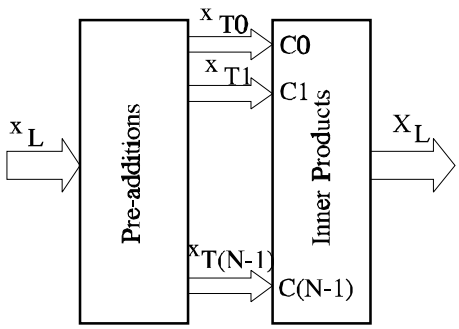

Figure 3.4.8: The computational steps of the algorithm

The first two level of the butterfly stages are simple and regular and common for all [Tx] x=0,1,...N-1 matrices. The $[L_N]$ NxN size butterfly matrix can be expressed in terms of the identity matrix $[I_{N/2}]$ and the opposite identity matrix $[I^*_{N/2}]$.

If the temporary data vectors between the levels of butterflies are indexed by the letters of B, C, D, E, F, then

$$[L_N] = \begin{bmatrix} [I_{N/2}] & [I^*_{N/2}] \\ [I^*_{N/2}] & -[I_{N/2}] \end{bmatrix} \qquad \textbf{(3.4.28)}$$

$$X_B = [A]\, x_L \quad ; \quad X_C = [B]\, X_B \quad ; \quad X_D = [C]\, X_C$$

$$[A] = [L_{64}]$$

$$[B] = diag\, scalesym150\{[L_8],[L_8],.....[L_8],[L_8] scalesym150\}$$

$$[C] = diag\, scalesym150\{[L_4],[L_4],[L_4],.....[L_4],[L_4],[L_4],[L_4] scalesym150$$

(3.4.29)

The signal flowgraph of the pre-addition phase for the following levels are shown in Fig. 3.4.2. and Fig. 3.4.3. for the even and the odd [Tx] matrices, respectively. On the figures the subtractions are not designated explicitly, but the lower horizontal line carries always the subtrahend. The flowgraphs show the appropriate connection points at the C and D level for these unique and the calculated $X_{Tx(i)}$ output signals of the pre-addition phase. It can be seen that many of the rows of the [Tx] matrices are share the same pattern, and the number of the different combinations are only N^2. The execution of the full pre-addition flowgraph for the 8x8 transformation needs 368 addition operation. The Tx(i) i=0,1,...N-1 notation at the output points of the pre-addition butterflies indicates the necessary inner product operations. To get the i-th transform domain data every Tx(i) combination of the input signals should be multiplied by the appropriate Cx value in the inner product processors, as given in Eq. 3.4.10. The architecture of the necessary inner product processors are discussed in the following section.

3.4.3 Internal structure of the inner product processors

Parallel hardware implementation of the direct 2-D transformation needs efficient arithmetic elements. To hold the hardware complexity at a reasonable level, the globally parallel, locally serial implementation style should be selected. This method has two advantage. The first is the simplicity of the basic serial arithmetic units, the second is the very low communication cost. Both of these are important to reduce the necessary resources.

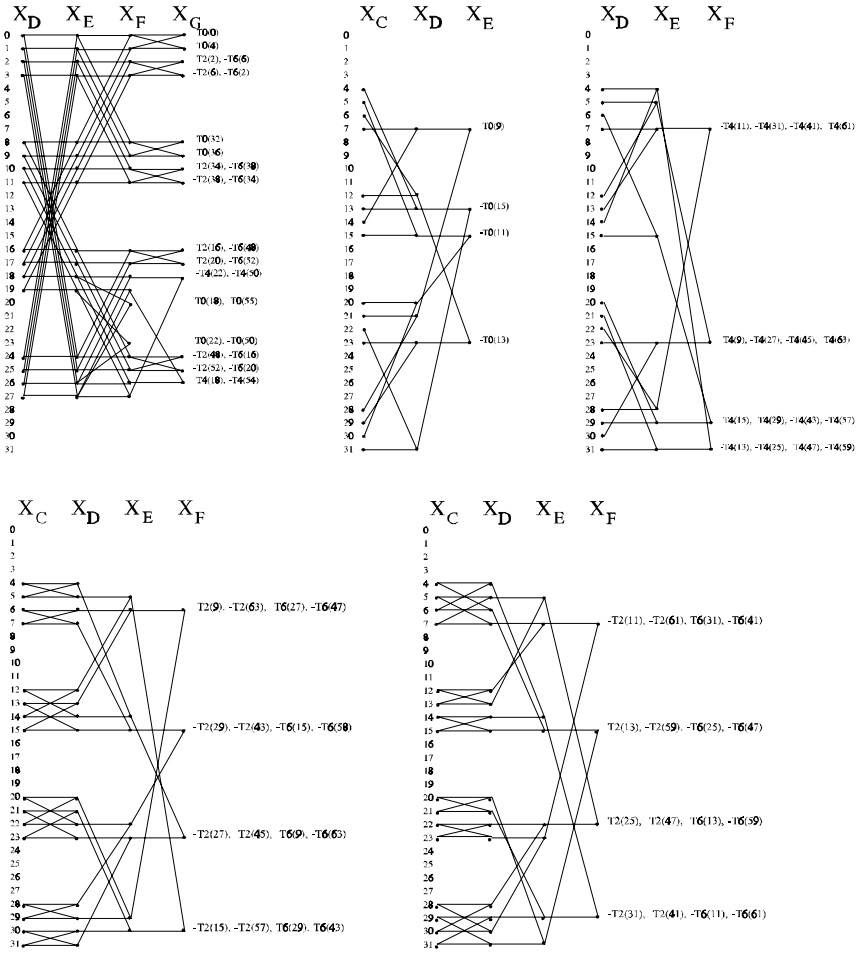

Figure 3.4.9:. The signal flowgraphs of the even T matrices

There are different methods to realize the inner product processors. The most widely used arrangement in the DCT processors is the memory

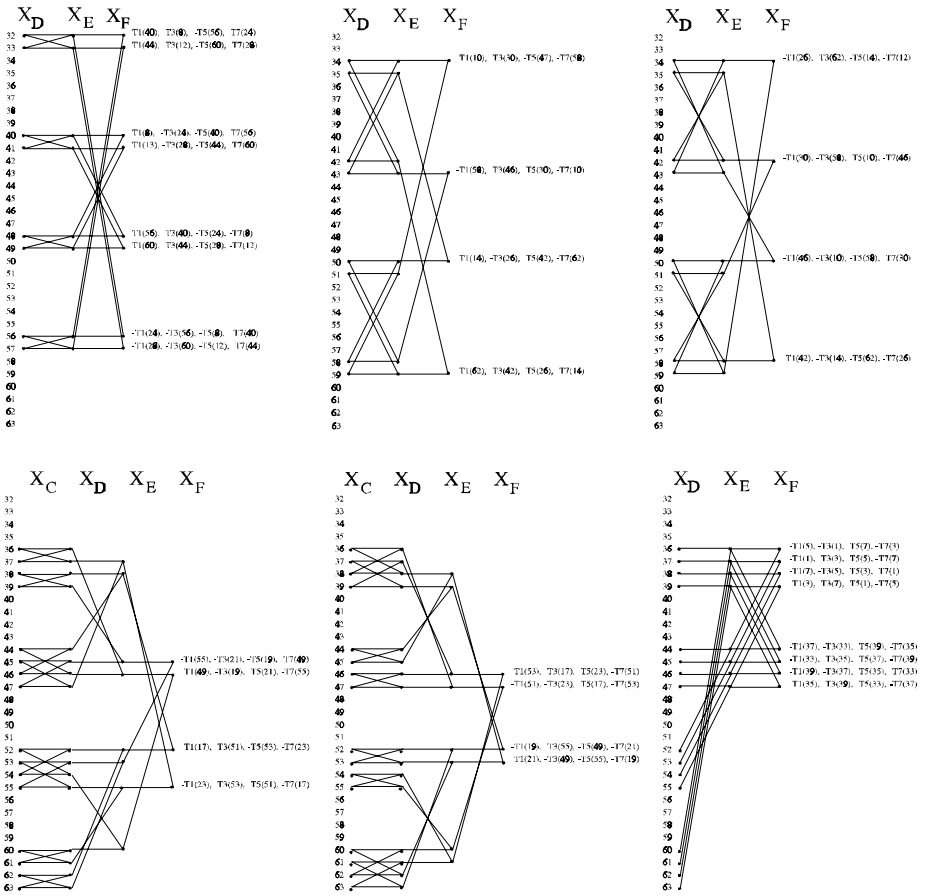

Figure 3.4.10:. The signal flowgraphs of the odd T matrices

look-up table based vector multiplier [Rao90]. This processors contains

memories to store the precomputed partial product sums and use parallel arithmetic units internally for the accumulation. The are some drawbacks of this arrangement. In some technology memory modules are difficult to realize (for example in FPGA devices), and the parallel arithmetic elements needs long basic execution cycles.

Recently a new efficient inner product processor arrangement have been proposed for the realization of fixed coefficient vector multiplications by serial arithmetic [Feh93]. This arrangement contains basic arithmetic primitives only, all of the operations are realized by pipe-lined serial adders, subtractors. Efficiency is achieved by using the Canonical Signed Digit code (CSD) of the fixed coefficients of the operations and by exploiting the available digit pattern coincidences between the coefficients CSD codes. The amount of reduction is depend on the size of the inner product operation, the actual numerical values and the CSD codes of the coefficients. Typically the method provide more than 30% reduction in the number of arithmetic elements and more than 70% in the number of the delay elements [Feh93].

The maximum size of the inner products necessary to calculate the X_L transformed domain coefficients of the 8x8 2-D DCT are 4. The multiplication factors are the odd or even elements of the Cx, x=0,1,...N-1 series, where Cx is defined by Eq. 3.4.9. The numerical values and the CSD codes of the coefficients of the 2-D DCT are given in Eq. 3.4.12., and internal structure of the inner product processors are shown in Fig. 3.4.4.

$$
\begin{array}{lcl}
C0 = 1_{10} & = & 1.0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
C1 = 0.9808_{10} & = & 1.0\ 0\ 0\ 0\ 0\ -1\ 0\ -1\ 0\ 0\ 0\ 1\ 0 \\
C2 = 0.9239_{10} & = & 1.0\ 0\ 0\ -1\ 0\ -1\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \\
C3 = 0.8315_{10} & = & 1.0\ -1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ -1\ 0\ 0 \\
C4 = 0.7071_{10} & = & 1.0\ -1\ 0\ -1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0 \\
C5 = 0.5556_{10} & = & 0.1\ 0\ 0\ 1\ 0\ 0\ -1\ 0\ 0\ 1\ 0\ 0\ -1 \\
C6 = 0.3827_{10} & = & 0.1\ 0\ -1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \\
C7 = 0.1951_{10} & = & 0.0\ 1\ 0\ -1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ -1\ 0
\end{array}
\qquad \textbf{(3.4.13)}
$$

The inner product processors assume 12 bit number representations of the Cx coefficients. Simulation results have shown, that it is satisfies the necessary computation accuracy.

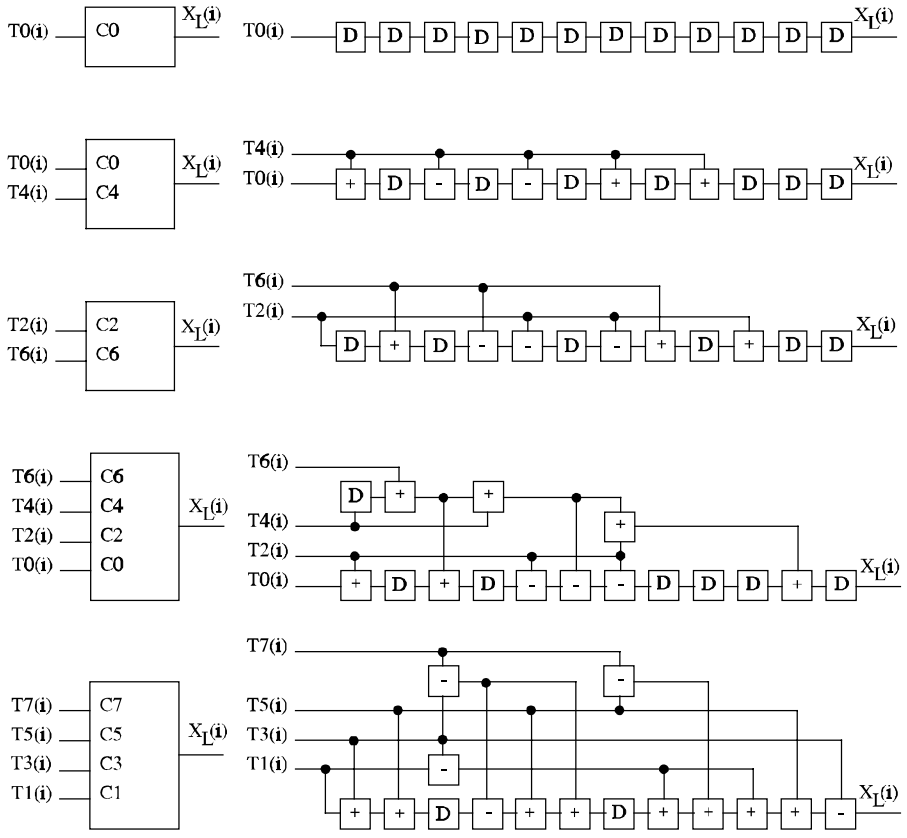

Figure 3.4.11:. The signal flow graph of the 8x8 2-D DCT inner product processors

3.4.4 FPGA IMPLEMENTATION OF THE 2D DCT ALGORITHM

Logic complexity of the available Field Programmable Gate Arrays (FPGAs) are relatively low for computation intensive DSP applications. Implementation of complex algorithms, such as the 2-D DCT on FPGA elements needs special precautions. The limitations of the available logic resources requires direct mapping of the arithmetic primitives to the FPGA basic logic cells. In case of serial arithmetic the optimal mapping is the one to one correspondence between the signal flow graph components and the hardware logic resources. Using SRAM based FPGAs, the relatively large

size of the basic logic cells are ideal to implement the pipe-lined serial arithmetic primitives. All the necessary logic are contained in one cell. This provides single level logic delays, high utilization of internal combinational and storage resources and minimum number of external connection points, reducing the routing complexity. As the routing resources of the FPGAs are designed for general usage of the high fan-in logic cells, in case of serial arithmetic most of the routing channels remains available to realize the complex topology of the signal flow graph of the given algorithm.

The hardware cost of the 2-D DCT is consist of two part. The first is the pre-addition module, the second is the inner product processor. The cost of the pre-addition module is proportional with the size of the transformation and is given by Eq. 3.4.14.

$$C_{PA} \cong N^2 \cdot \log_2 N^2 \qquad \textbf{(3.4.31)}$$

The cost of the inner product processors for the selected 12 bit precision can be calculated from Fig. 3.4.4. The processors of the X_L transformed domain data components are represents different complexity. The cost of all modules are summarized and listed in Table 1. It shows, that the total number of arithmetic primitives is less than 1000.

	Pre-add.	C1-C7 IPPs	C0 delay	C0,C4 IPPs	C2,C6 IPPs	C0-C6 IPPs	Total
+/-	368	416	0	20	48	144	996
Delay	0	64	48	28	48	112	300

The execution time of the computation using the conventional LSB first computation scheme is $B_x+B_c+\log_2 N^2$ bit clock cycle, where B_x and B_c are the number of bits in the input data and Cx coefficient representation. The available bit clock rate with the Xilinx 4000 series FPGAs is about 30-40 MHz. Assuming the usual B_x=9 and B_c=12 bit values, one transformation needs less than 30 clock cycle, so the available thoughput is 1 million transformation/sec.

Bibliography

[Ahm74] N. Ahmed, T. Natarajan, K. R. Rao," Discrete cosine transform," IEEE Trans. Computers, Vol. C-23, pp. 90-93, Jan. 1974.

[Ait91] D. Ait-Boudaoud, M. K. Ibrahim,"Efficient Implementation of Bit-Level Signal Processors", Proc. of IEEE ISCAS, 1991

[Bla85] R. E. Blahut, "Fast Algorithms for Digital Signal Processing," Addison-Wesley, 1985

[Bur77] C. S. Burrus,"Digital Filter Structures Described by Distributed Arithmetic," IEEE Trans. on Circuits and Systems, Vol. CAS-24, No.12, pp. 674-680. Dec. 1977.

[Cho91] N. I. Cho, S. U. Lee,"Fast algorithms and implementation of 2-D discrete cosine transform," IEEE Trans. Circuits and Systems, Vol. 38. pp. 297-305, March, 1991.

[Den85] P. Denyer, D. Renshaw, VLSI signal processing: A bit-serial Approach. Addison-Wesley, 1985

[Diet90] C. B. Dietrich, et. al.,"A Complete Audio Decoder IC for U.S. Television Stereo Using Serial Arithmetic", Proc. of IEEE CICC 1990

[Duh90] P. Duhamel, C. Guillemot,"Polynomial transform computation of 2-D DCT," in Proc. of IEEE ICASSP 1990. pp. 1515-1518. Apr. 1990.

[Feh91] B. Fehér,"Field-programmable gate array implementation of the recursive Walsh-Hadamard transformation," presented at the Int. Conf. on DSP Appl. and Techn. Berlin, Germany, Oct. 28-31, 1991.

[Feh92a] B., Feher, "Resonator Based Digital Filters Using Field Programmable Gate Array Elements, Proc. of IEEE ASIC,

1992.

[Feh92b] B., Fehér, T., Lázár, "Digital FIR Filter Structure with XILINX Programmable Logic Circuit", (in Hungarian) Mérés- és Automatika, Vol. 40. No. 5. pp. 294-298. Dec. 1992.

[Feh93a] B. Fehér," Efficient Synthesis of Distributed Vector Multipliers," Microprocessors and Microprogramming, Vol. 38. pp. 345-350, Sept, 1993

[Feh93b] B., Fehér, "Coefficient Dependent Synthesis of FIR Digital Filters", Proc. of the 4th Mideuropean Conference on Custom/Application Specific Integrated Circuits, 1993

[Feh94] B., Feher et. al. "Parallel Implementation of teh Statistic Feature method" Proceedings of uP94.The 8th Symposium on Microcomputer and Microprocessor Applications Budapest, October, 1994. pp.211-219.

[Hou87] H. S. Hou, "A fast recursive algorithms for computing the discrete cosine transform," IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-35, pp. 1455-1461, Oct. 1987.

[Hsi90] Hsieh, Hung-Cheng, et. al.,"Third-generation architecture boosts speed and density of field programmable gate arrays," IEEE Conference on Custom Integrated Circuits, 1990

[Hun90] H., Hung-Cheng, et. al.,"Third-generation architecture boosts speed and density of field programmable gate arrays," Proc. of IEEE CICC, 1990

[Kam82] F. A. Kamangar, K. R. Rao, "Fast algorithms for the 2-D discrete cosine transform," IEEE Trans. Computers, Vol. C-31, pp. 899-906, Sept, 1982.

[Lee84] B. G. Lee, "A new algorithm to compute the discrete cosine transform," IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-32, pp. 1243-1245, Dec. 1984.

[Liu93] K. J. R. Liu, C. T. Chiu,"Unified parallel structures for time-recursive discrete cosine/sine/Hartley transform," IEEE Trans.

on Signal Processing, Vol. 41. No. 3. pp. 1357-1377. March 1993.

[Lyo76] R. F. Lyon,"Two's complement pipeline multipliers," IEEE Trans. Communications, pp. 418-425, Apr. 1976.

[Mas91] D. L. Maskell, "Hardware Implementations for Digital Filters with Low Coefficient Sensitivity," IEEE Int. Symp. on CAS, 1991. vol. 1, pp.504-507.

[Nar78] N. J. Narashimha, A. M. Peterson, "On the computation of the discrete cosine transform," IEEE Trans. Commun. Vol. COM-26, pp. 934-936, June 1978.

[Pec86] G. Péceli,"A common structure for recursive discrete trans forms," IEEE Trans. Circuits Syst., vol. CAS-33 no. 10. pp. 1035-1036, Oct. 1986.

[Pec89] G. Péceli,"Resonator-Based Digital Filters," IEEE Trans. Circuits Syst., vol. CAS-36, no. 1. pp. 156-159, Jan 1989.

[Pec90] G. Péceli, B. Fehér,"Digital filters based on recursive Walsh-Hadamard transformation," IEEE Trans. Circuits Syst., vol. CAS-37, no. 1. pp. 150-152, Jan. 1990.

[Pel74] A. Peled, B. Liu,"A new hardware realization of digital filters," IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-22, no. 6, pp. 456-462. Dec. 1974.

[Rao90] K. R. Rao, P. Yip,"Discrete cosine transform," Chapter 5., Academic Press, San Diego, 1990.

[Rei60] G. W. Reitwiesner, "Binary Arithmetic. Advances in Computers," vol. 1, pp. 232-308. Academic Press, 1960.

[Vet85] Vetterli, M: Fast 2-D Discrete Cosine Transform. Proc. of IEEE ICASSP, 1985.

[Xil91] Xilinx XNF Netlist Specification, Xilinx Inc., Version 4.00a-10/18/1991.